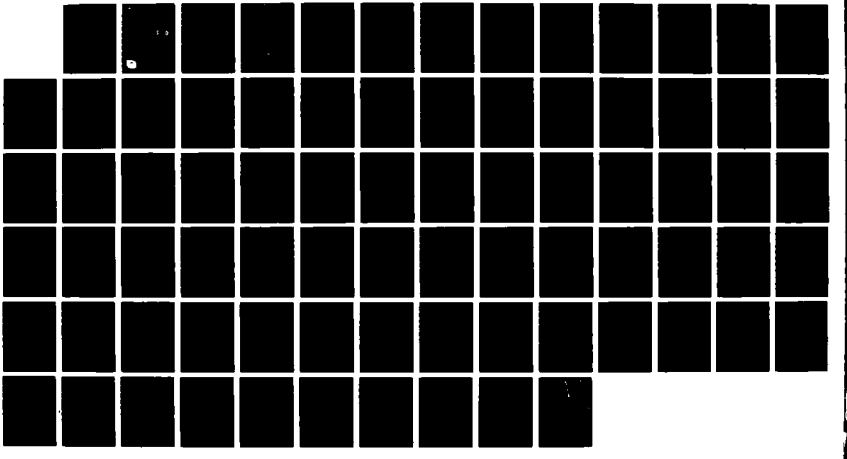
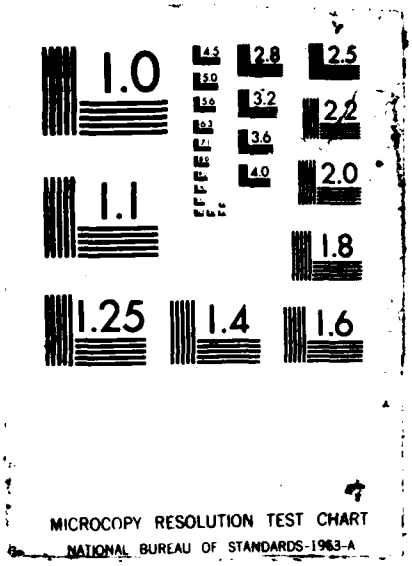
AD-A185 664 INFORMATION INTERFACE RELATED SOURCES SEE-INFO-003-0010 1/1
(SOFTWARE ENGINEE. (U) INSTITUTE FOR DEFENSE ANALYSES
ALEXANDRIA VA R P MORTON ET AL. APR 85 IDA-P-1821
UNCLASSIFIED IDA/HQ-85-29655 MDA903-84-C-0031 F/G 12/5 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD-A185 664

IDA PAPER P-1821

INFORMATION INTERFACE RELATED SOURCES SEE-INFO-003-001.0

Richard P. Morton
Jack C. Wileden

April 1985

Prepared for

Office of the Under Secretary of Defense for Research and Engineering

INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street, Alexandria, VA 22311

IDA Log No. HQ 85-29655

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Public release/unlimited distribution.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) P-1821	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Institute for Defense Analyses 6b OFFICE SYMBOL IDA	7a NAME OF MONITORING ORGANIZATION
6c ADDRESS (City, State, and Zip Code) 1801 N. Beauregard St. Alexandria, VA 22311	7b ADDRESS (City, State, and Zip Code)
8a NAME OF FUNDING/SPONSORING ORGANIZATION STARS Joint Program Office 8b OFFICE SYMBOL (If applicable) SJPO	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER MDA 903 84 C 0031
8c ADDRESS (City, State, and Zip Code) 1211 Fern St., C-107 Arlington, VA 22202	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. T-D5-429 / WORK UNIT ACCESSION NO.
11 TITLE (Include Security Classification) Information Interface Related Sources SEE-INFO-003-001.0 (U)	
12 PERSONAL AUTHOR(S) Richard P. Morton, Jack C. Wileden	
13a TYPE OF REPORT Final 13b TIME COVERED FROM _____ TO _____ 14 DATE OF REPORT (Year, Month, Day) 1985 April 15 PAGE COUNT 84	
16 SUPPLEMENTARY NOTATION	

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Information Interfaces; Software Engineering Environment; Software Tools.

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

This document is a compendium of the states of the art and practice related to information interfaces in environments and non-DoD related software work products. It characterizes information interfaces in software engineering environments (SEEs), describes the information interfaces in a select sample of environments and tools, surveys existing information interface technology, and points the way to additional literature from which a more complete treatment of information interfaces and information interface technology may be obtained.

Appendix 1 contains a paper, The TRW Software Productivity System, by Ann B. Marmor-Squires, and Appendix 2, Some Thoughts on a Taxonomy for Software Engineering Objects by Leon G. Stucki.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ■ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL	22b TELEPHONE (Include Area Code) / 22c OFFICE SYMBOL

IDA PAPER P-1821

INFORMATION INTERFACE RELATED SOURCES SEE-INFO-003-001.0

Richard P. Morton
Jack C. Wileden

April 1985

INSTITUTE FOR DEFENSE ANALYSES

Contract No. MDA 903 84 C 0031
Task T-D5-429

Preface

This is a compendium of the state of the art and practice related to information interfaces in environments and non-DoD related software work products. It characterizes information interfaces in software engineering environments, describes the information interfaces in a select sample of environments and tools, surveys existing information interface technology, and points the way to additional literature from which a more complete treatment of information interfaces and information interface technology may be obtained.

This compendium was developed with the perspective of trying to be as helpful as possible to the people who have to perform the follow-up tasks. As a result, breadth was given priority over depth in the literature search. The intention was to try to identify as many areas of interest as possible, even though resources permitted only a superficial analysis of each area. The requisite in-depth analysis is one of the recommended follow-up tasks.

The compendium is divided into sections reflecting development activities (sections 2-6), management and support activities (sections 7-12), technologies (sections 13-17), and application areas (section 18). In general, a brief overview begins each section. A list of references pertaining to the section topic concludes each section. Some of the sections also contain a more detailed review of one or more tools or environments relevant to the section topic. These reviews are intended to provide examples of notable information interfaces and/or information interface technology related to the topic of the section. Conclusions and recommendations conclude the compendium.

Acknowledgement

The authors would like to gratefully acknowledge the contributions of many reviewers, including the JSSEE team, Gill Berglass, Ann Marmor-Squires, Leon Stucki and especially Sam Redwine. Sarah Nash contributed significantly in several ways, particularly in performing the electronic literature searches; Cynthia Hillman helped with the editing and production details; Joyce Walker and several other IDA secretaries assisted with typing the references and drawing the figure. Their efforts are also greatly appreciated.

Table of Contents

Information Interfaces

1.0 Introduction

1.1 Background

The Software Technology for Adaptable Reliable Systems Joint Program Office (STARS JPO) was established by the Deputy Under Secretary of Defense for Research and Engineering (DUSDRE) for Research and Advanced Technology (R&AT) to plan and implement a program to improve the state of practice of software technology throughout the DoD. This program has as one of its central efforts the development of a Joint Service Software Engineering Environment (JSSEE).

A goal of the JSSEE development is to establish an architecture which will accommodate the incorporation of new software technology in an evolutionary manner. This goal imposes stringent requirements on the software architecture. More importantly here, a goal to permit migration of project data to other environments raises the issue of information interfaces to primary importance.

As defined in the IEEE Standard Glossary of Software Engineering Terminology, an interface is a shared boundary. An information interface is an inter-environment, inter-tool or user-tool interface that consists of data (as opposed to invocation or control interfaces). These boundaries are specified using an information interface specification. The data that passes across such a boundary is expected to conform to the specification for the interface. [A glossary of terms related to information interfaces is under development. The definitions used here conform to an early draft of that glossary.]

The purpose of this task is to provide the STARS JSSEE effort with a technical compendium of the state of the art and practice related to information interfaces in environments and non-DoD related software work products. The ultimate purpose for studying information interfaces is to be able to design the JSSEE in a way that makes it useful in a wide variety of project environments, using a variety of methods, techniques, tools, notations, and user interfaces, including many not yet developed. To do this it is important to understand the information interfaces that exist in a generic sense in software development and support, how they are used by the different methods and tools, and how they are represented in the different formats and notations. It is also important to understand information interface technology, that is, techniques for specifying information interfaces and methods for using, organizing, controlling and assessing them.

The purpose of this report is to characterize the concept of Information Interfaces in software engineering environments (SEEs), to describe the information interfaces present in a select sample of environments and tools, and to point the way to additional literature from which a more complete treatment of information interfaces and information interface technology may be obtained.

1.2 Overview and Scope

While the concept of a JSSEE encompasses the total working environment of a software engineer, with respect to information interfaces we limit our concern to that part of the environment that is potentially automated. Such an automated environment (referred to in the remainder of this report as an environment) consists of a collection of tools that operate on information fragments.

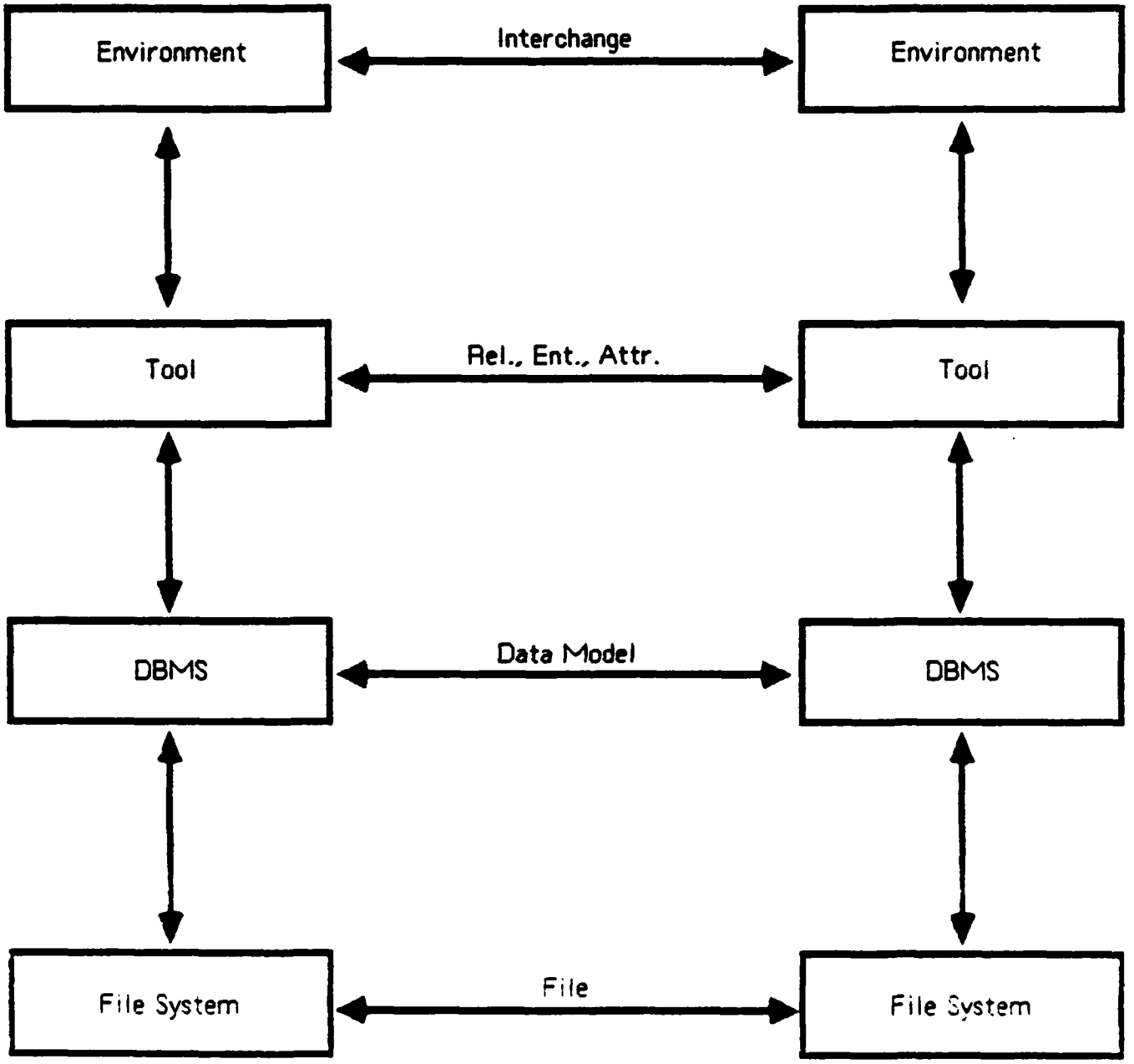
Within this environment the information interface specifications specify information fragments manipulated by the tools. There are several ways of categorizing these information fragments. Some information fragments are external and some are internal. External information fragments are user-environment information fragments or environment-environment information fragments. User-environement information fragments are ones that need to be intelligible to people as well as tools. Documentation, analysis results, source code and review reports are examples of external information fragments. Internal information fragments are ones that are produced by one tool for the consumption of other tools. A parse tree would normally fall into this category.

Some information fragments are root fragments and some are non-root fragments. Root fragments are fragments that conform to complete information interface specifications, such as a specification or symbol table. Non-root fragments are meaningful parts of those complete fragments. Some examples of non-root fragments are functions, attributes and priorities of a specification, and symbols, types and other attributes of a symbol table. Some non-root fragments are relationships between other non-root or root fragments, such as the traceability relationships between a requirements document and a design document, or the nesting structure of a symbol table.

The specifications of (root, non-root) information fragments are (root, non-root) information interface specification fragments. That is, an important aspect of the notion of a fragment is its relationship to an information interface. Fragments are not arbitrary pieces of specifications or data.

The above definitions characterize information interfaces in a few ways that are used in the remainder of this paper. There are other ways of characterizing information interfaces. Consider the diagram in Figure 1. The vertical arrows represent invocation

Figure 1. One structure for information interfaces



and control interfaces (which may also be partially information interfaces), and the horizontal arrows represent information interfaces between like components. From this it can be seen that specifications of interfaces need to define files (the physical models of data storage), logical data models used in data base management systems, the specific relations, entity types and attributes of the data used by specific tools (semantic data models), and interchange formats used for delivering the output of one environment to another. This hierarchy, while somewhat similar to the hierarchy of fragments described above, takes the perspective of a data base designer or administrator.

The developers of the JSSEE may need to take another perspective, one closer to the data itself. In this case the types of data fragments can be categorized in several different ways. One such way is in terms of the nature of the processing to be done on them. Here, there would be sequential items (perhaps called files), structured storage items (perhaps relations), graphical items, screens, binary data, etc. Another way to characterize the data fragments is in terms of the way they are used, for example, as documentation text files, source code text files, load modules, forms, input edit templates, help messages, test data, measurement information, accounting information, etc. A specific example of how this might be done is shown in Appendix 2.

Since the intent of the JSSEE is to automate as much of the software engineering process as is practical within the state of the art, we have had to consider information interfaces from all aspects of software development, maintenance and management. We categorize these aspects as follows:

* Phases (requirements, design, coding, etc.)

* Management and support activities (project management, configuration management, training, etc.)

* Technologies (artificial intelligence, graphics, etc.)

* Applications (real-time, command and control, etc.)

Phases and management and support activities have traditionally been the subject of software engineering research. While there is much dispute over what phases and management and support activities software projects should employ, and indeed, over whether the traditional model of phases is even appropriate, for this task we have been instructed to structure this report in concert with the draft DOD-STD-2167, Defense System Software Development.

Technologies and applications are frequently (but not always) ignored in discussions of environments. Technologies, such as graphics, are extremely general topics whose scopes exceed their applicability to software engineering environments. Our interest,

therefore, is limited to what they contribute to software engineering and to the design of environments. This can be viewed as what implications the architectural choices for the design of the JSSEE have for information interfaces. For example, if JSSEE workstations are graphics oriented, then graphics related information interfaces need to be part of the JSSEE. If the JSSEE is implemented as a distributed system, then networking interfaces are relevant.

In addition to the technologies we discuss below, there are several others that could also be listed or covered more extensively but are not because there are other JSSEE related reports due that cover these areas. These include human engineering, multilevel security, standards and database management.

In general, we have limited our search for application specific information interfaces to those we believe to be highly likely applications to be developed with a JSSEE. These include real-time applications and some of the problems dealt with in command and control and decision support systems. Many of the subjects listed as technologies to be used in building the JSSEE are also technologies to be used in the MCCR products to be built with the JSSEE, and, therefore, are application related issues. The topics of human engineering, security and database management are mentioned briefly in this context.

To date, there is little literature that discusses information interfaces per se. To learn about information interfaces one frequently needs to read between the lines in descriptions of languages, tools or methods in order to identify the classes of objects the author is describing. For example, in reviewing a description of a formal specification notation, one might notice that the notation represents objects such as functions, relations, sets, variables, operations on sets and operations on functions. These classes of objects are the kinds of information that a user of that notation needs to deal with, and the kinds of information that might become part of an information interface specification between, say, a specifications editor and a specifications analyzer.

Because of the need to cover a wide area in a relatively brief time, the search for information about information interfaces has been limited to sources that were easily available. Undoubtedly, the reader will find one or more of his favorite tools or references missing from this list. Suggestions will be appreciated since the work of defining the information interfaces of the JSSEE is still to come.

The literature search consisted of reviews of a limited set of recent journals plus an on-line search of several technical databases using numerous keyword combinations. The on-line search yielded over 300 interesting sounding titles, so abstracts for these were also reviewed. This still netted over 100 documents of

interest, far too many to actually read in the available time. Consequently, many of the citations listed here cannot be interpreted as recommended reading, only likely places to look for useful information. Over 50 documents were actually read or scanned. These gave us many insights into what information interfaces exist in various systems, as well as insights into how to look for information interfaces. The citations taken from abstracts, therefore, reflect our best judgement regarding where promising sources can be found.

What we learned about looking for information interfaces is that the easiest places to find them in the literature are in descriptions of methods and languages. In most cases descriptions of methods are concerned with identifying the objects that someone carrying out the method will need to assemble and produce. Similarly, language descriptions identify classes of objects within the language (statements, expressions, etc.), as well as objects manipulated by the language (files, variables, etc.).

Most of the subject headings we have chosen for this report are not sufficiently well standardized in their usage to be clearly bounded. Therefore, the reader will find numerous cross references within the text below. It is hoped that these will both prompt the reader to look in adjoining subjects for relevant information, and remind him that significant overlap can be found along the boundaries between phases, particularly because of the multiple viewpoints that apply to such situations; the output from one activity is the input to the next.

1.3 Report Organization

Most of the remainder of this report is divided into sections reflecting phases (sections 2-6), management and support activities (sections 7-12), technologies (sections 13-17), and application areas (section 18). In general, each of these sections begins with a brief overview and concludes with a list of references related to the topic of that section. Some of the sections also contain a more detailed review of one or more tools or environments relevant to that topic. These more detailed reviews are primarily intended to provide examples of notable information interfaces and/or information interface technology pertaining to the topic of the section. Conclusions and recommendations complete the report.

1.4 General References

The references in this section cover more general subjects than each of the remaining sections of this report. The following document describes the intent of the JSSEE from a user's perspective.

Operational Concept Document (Draft), STARS Joint Program Office, JSSEE-OCD-000.2, November 15, 1984.

The following references relate to the structure of a software engineering environment database. They give some perspective on relationships that might not be found elsewhere.

Santoni, Patricia A., The Project Development Data Base: The Core of an Automated Software Engineering Environment, Naval Ocean Systems Center, Technical Note 932, 24 October 1980.

Wassermann, Anthony I.; Prenner, Charles J., "Toward a unified view of data base management, programming languages and operating systems - a tutorial", Information Systems, Vol. 4, No. 2, 1979, pp. 119-126, Pergamon Press, Ltd.

The following reference discusses information interfaces from a market perspective.

Redwine, Samuel T., Jr., "The Future Government and Industry Software Tools Marketplace", Proc. Washington Ada Symposium, March 25-27, 1984, Washington D.C. Chapter Ada Technical Committee and Johns Hopkins University.

Probably the most common form of information interface technology (that is, technology used specifically for defining information interface specifications) are data dictionary/directory systems. Most data dictionary references are product specific vendor literature; other discussions of the use of data dictionaries can be found in some of the database management texts. The following are additional references.

Lefkovits, Henry C., Sibley, Edgar H., Lefkovits, Sandra L., Information Resource - Data Dictionary Systems, QED Information Sciences, 1983.

Leong-Hong, Belkis W., Plagman, Bernard K., Data Dictionary Directory Systems: Administration, Implementation and Usage, Wiley & Sons, 1982.

Marti, Robert W., "Integrated Database and Program Descriptions using an ER-Data Dictionary", Entity Relationship Approach to Software Engineering, North-Holland, 1983.

In addition to the specific tools and environments discussed in the sections below, there are, of course, a very large number of other tools and environments, many of which may become part of the JSSEE. There are far too many to cite individually (for example, the IITRI listing contains the names of several hundred tools) so we give here a few references to collections of citations for tools and environments literature. It should be kept in mind that published articles about tools are not usually detailed enough to provide a very complete list of the information interfaces related to such tools and environments. The best

sources are the user manuals and system documentation about the tools and environments. The published literature simply provides pointers to who is building what and can provide some understanding of such issues as completion and comprehensiveness of the tool and environment development.

Henderson, Peter, Editor, Proc ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Environments, ACM, 1984.

Houghton, Raymond C., Jr., ed., Proceedings of the NBS/IEEE/ACM Software Tool Fair, National Bureau of Standards, NBS Special Publication 500-80, 1981.

IIT Research Institute, Data and Analysis Center for Software, Listing of the Software Tool Information Database, January, 1985.

Miller, Edward, Tutorial: Automated Tools for Software Engineering, IEEE Computer Society, 1979.

Stucki, Leon, and Houghton, Raymond C., Jr., eds., Proceedings: SOFTFAIR 83, the First Conference on Software Development Tools, Techniques, and Alternatives, IEEE Computer Society Press, 1983.

Wasserman, Anthony I., Tutorial: Software Development Environments, IEEE Computer Society, 1981.

Current publication that containing frequent references to tools and environments are:

ACM SIGSoft Software Engineering Notes (see especially the Abstracts)

IEEE Computer Society Computer

IEEE Tansactions on Software Engineering

The Proceedings of the periodic International Conference on Software Engineering

The Proceedings of the annual IEEE Computer Society Computer Software and Applications Conference

There are five environments which have been or are currently being developed within the DoD. These will have special significance for the JSSEE since they are expected to become important environments within the DoD contracting community, and are intended to be on the migration path to the JSSEE. References are:

"Ada Language System Specification", U. S. Army Communication Electronics Command, November 1983.

"Distributed Computing Design System (DCDS) Methodology Capability Demonstration", CDRL C004, TRW Report No. 38392-G950-041, October 1984.

"Facility for Automated Software Production (FASP) Handbook, Revision 1.0", Naval Air Development Center, 24 March 1980.

"System Specification for the Ada Integrated Environment, Intermetrics Inc., Report No. AIE(1).

"System Specification for the Ada Language System/Navy, Naval Ocean Systems Center, 12 September 1983.

The following are general texts on the subject of software engineering, tools and environments.

Bauer, F.L., editor, Software Engineering; an Advanced Course, Springer-Verlag, 1977.

Blank, J., and Krijger, M.J., eds., Software Engineering Methods and Techniques, Wiley-Interscience, 1983.

Hunke, Horst, ed., Software Engineering Environments, Proceedings of the Symposium held in Lahnstein, Germany, 16-20 June 1983.

Jensen, R.W., and Tonies, C.C., Software Engineering, Prentice-Hall, 1979.

Fairley, R.E., Software Engineering Concepts, McGraw-Hill, 1984.

Pressman, R.S., Software Engineering: A Practitioner's Approach, McGraw-Hill, 1982.

Riddle, W.E., and Fairley, R.E., Software Development Tools, Proceedings of a Workshop held at Pingree Park, Colorado, May 1979, Springer-Verlag, 1980.

Shooman, M.L., Software Engineering: Reliability, Development and Management, McGraw-Hill, 1983.

2.0 Requirements and Specifications

2.1 Two views of requirements

The software industry uses two somewhat different definitions of the terms "requirements" and "requirements analysis". Most of the embedded systems community treats software requirements as a refinement of systems requirements. In this context, by the time the software analyst gets involved, many of the software requirements have already been specified. In fact, the job of the

analyst is to insure that the stated requirements are complete and consistent. In contrast, the information systems community typically views the software requirements as initially unknown, and it is the job of the analyst, working with the customer or user to "discover" the requirements. The significance of this difference in views is that different researchers have taken different approaches in providing languages, tools or methods for performing the requirements activities of software development. The reader should keep these two views in mind when reading the literature on requirements specification and analysis.

2.2 Tools Review

2.2.1 SREM

The Software Requirements Engineering Methodology (SREM) is one component of the Distributed Computing Design System (DCDS) developed by TRW for the US Army Ballistic Missle Defense Advanced Technology Center. SREM and DCDS are efforts to produce a unified methodology and environment supporting all stages of the software development process. Since one of the stated objectives of DCDS is to employ a uniform underlying model in the tools supporting each stage of development, the tools and information interfaces associated with the various individual components of DCDS are all quite similar. Since SREM is the oldest, most mature and most representative of the DCDS tools, as well as one of the few requirements tools in existence, we have chosen to consider only SREM in this compendium. In particular, we do not mention the DCDS tools supporting design or testing activities in the appropriate sections, since they do not differ enough from SREM or from the other tools already considered in those sections to make their addition there essential.

The external information interface provided by SREM is RSL, the Requirements Statement Language. RSL permits a user to describe the components needed in a software system and the interrelationships among those components. In essence, RSL offers a relational database into which users may enter information about the properties that a software system is intended to have. An interesting and valuable feature of RSL is the fact that it is extensible. Arbitrary new categories of components and interrelationships among components may be added to RSL by a user. SREM also provides consistency checking tools to help RSL users in assessing the quality of the RSL requirements documents that they generate.

The internal information interface provided by SREM is an element- attribute-relation structure encoding the information from RSL descriptions. This internal information interface is uniformly used throughout DCDS and is processed by all DCDS tools. In fact, the same tools are used in various stages of development by DCDS, with table-driven interpretation to enforce the appropriate semantics depending upon which stage of development is currently being supported. Thus the tables that control the

interpretation by DCDS tools are also an internal information interface.

A reference on SREM is:

SREM User's Manual, SREM Final Report, Volume II, CDRL C005, TRW Report No. 27332-692L-126.

2.3 Literature

It should be kept in mind that there is no agreement on the placement of the line between requirements and design. The reader may find that some of the work cited in Section 3 below falls within his perspective of requirements, or conversely, that some of the material in this section belongs in Section 3.

Additionally, much of the future paradigms work (Section 16) is based on the notion that a formal approach to requirements specification will eventually lead to an automated way of producing the production system. Consequently, considerable research in requirements can be found in the future paradigms section.

The requirements analysis activities normally end with the production of a requirements analysis document. Other root information fragments frequently associated with this phase, either as inputs or outputs, include a concept definition document, feasibility study, system specification and acceptance test plan. Non-root information fragments usually found in the requirements analysis activity include functions, performance specifications (meaning response times, data capacities, etc.) and acceptance criteria. The documents listed below generally fall into this pattern, but they present alternative approaches to documenting each of these information fragments. The ones that present formal approaches to specifications typically add the objects commonly found in some branch of mathematics, such as sets, elements and functions. Documents that describe specific automated systems for capturing requirements typically add some structuring relationships and a set of attributes to the list of non-root fragments.

Babb, Robert G., and Mili, Ali, eds., Workshop Notes: International Workshop on Models and Languages for Software Specification and Design, March 30, 1984, Orlando, Cite Universitaire, Quebec, DIUL-RR-8408.

Beichter, F., Hertzog, O., Petzsch, SLAN-4 Reference Manual and Design Rationale, Tech. Rep. GTR 05.272, IBM Laboratory Boeblingen, 1982.

Borgida, Alexander, "Features of Languages for the Development of Information Systems at the Conceptual Level", IEEE Software, V.2, No. 1, January 1985, pp. 63-72.

DeWolf, J. B., "Requirements Specification and Preliminary Design for Real-Time Systems", Charles Draper Lab., Inc., Cambridge, MA, The IEEE Computer's Society's First International Computer Software and Applications Conference, 8-11 Nov. 1977, Chicago, IL. Publ. IEEE.

Forsyth, D. Y.; Ward, A. O., SAFRA: Controlled Requirements Expression, British Aerospace Aircraft Group, Preston, (England).

Heninger, K.L., "Specifying Software Requirements for Complex Systems: New Techniques and Their Application," IEEE Transactions on Software Engineering, Vol. SE-6, No. 1, Jan. 1980, pp. 2-13.

An Introduction to SADT Structured Analysis and Design Technique, SofTech Inc., Document No. 9022-78R, November 1976.

Jackson, M.A., System Development, Prentice-Hall, 1983.

Lamb, S.S.; Leck, V.G.; Peters, L.J. and Smith, G.L., "SAMM: A Modeling Tool for Requirements and Design Specification", Proceedings of COMPSAC 78, IEEE Computer Society, 1978, pp. 101-08

Miyamoto, I.; Yeh, R. T., "A Software Requirements Analysis and Definition Methodology for Business Data Processing", Univ. of Maryland, College Park, MD, AFIPS Conference Proceedings, V. 50, 1981 National Computer Conference 571-81,4-7 May 1981, Chicago, IL

Meyer, Bertrand, "On Formalism in Specifications", IEEE Software, V.2, No. 1, January 1985, pp. 6-26.

Shaw, R.C., Hudson, P.N., Davis, N.W., "Introduction of a Formal Technique into a Software Development Environment (Early Observations)", Software Eng. Notes, V9, No. 2, Apr. 1984. ACM SIGSOFT

Stinson, Susan K.; Wasserman, Anthony I., "Specification Method for Interactive Medical Information Systems", Univ. of Calif., San Francisco, Proc. Annu. Symp. Comput. Appl. Med. Care 4th, Proc. of the Annu. Conf. of the Soc. for Adv. Med. Syst., 12th V. 3, Washington, DC, Nov. 1-5, 1980. Publ. by IEEE, Piscataway, NJ, 1980 p. 1471-1478.

Tausworthe, R. C., Preparation Guide for Class B Software Specification Documents, Jet Propulsion Lab., Report No.: NASA-CR-162431; JPL-PUB-79-56, 1 Oct 79.

Teichroew, D. and Hershey, E.A., III: "PSL/PSA: A Computer Aided Technique for Structured Documentation and Analysis of Information Processing Systems," IEEE Trans. on Software

Engineering, Jan., 1977.

Zave, Pamela, "An Overview of the PAISLey Project - 1984," Software Engineering Notes, V. 9, No. 4, July, 1984, ACM SIGSOFT.

3.0 Preliminary and Detailed Design

Even though DOD-STD-SDS lists preliminary design and detailed design as separate phases in the life-cycle, we have grouped them here because most of the research in software design does not distinguish between them.

3.1 Tools Review

3.1.1 Arcturus

The Arcturus system [Standish, 1983], [Standish and Taylor, 1984], developed at the University of California at Irvine, provides a variety of interesting information interfaces. Arcturus was an experiment in supporting highly interactive creation and debugging of Ada programs. Its style and user interface were modelled on the InterLisp environment. Among its information interfaces is an interactive Ada program design language (PDL), which will be described here. Other Arcturus information interfaces are considered in section 4.1.1.

The Arcturus PDL provides designers of Ada programs with an information interface suitable for detailed design. The PDL consists of a subset of the Ada language (see 4.1.1 for details on the composition of that subset), expressed in standard Ada syntax, augmented by a mechanism for including pseudo-code descriptions of data objects and operations in a design. Specifically, the Arcturus PDL permits an arbitrary string of characters enclosed by "{" and "}" to appear anywhere that a name, type, declaration, expression or statement could appear in a normal Ada program. Thus a designer is free to express more abstract descriptions of parts of an eventual Ada program than would be possible using the Ada language alone. For example, a designer might write

```
AvgReading := {average of last ten samples};
```

or

```
{determine next anticipated position};
```

as part of an Arcturus PDL design.

The Arcturus PDL represents a typical program design language, similar in spirit to PDLs dating back to Caine and Gordon [Caine and Gordon, 1975]. It is singled out for treatment here partially due to its applicability to Ada and partially due to its relationship to other information interfaces and tools in

the Arcturus environment. In particular, the Arcturus PDL can be refined stepwise into executable Ada by expanding the user-definable roots represented by the strings enclosed by "{" and "}". The Arcturus syntax analyzer, pretty printer and template-driven editor can also all be applied to PDL descriptions. Finally, Arcturus PDL can be transformed into the same internal representation, itself an information interface, as can Ada programs written in the Arcturus subset of Ada. Both that subset and the internal representation are described in section 4.1.1.

References related to Arcturus are:

Standish, Thomas A., "Interactive Ada in the Arcturus Environment", Ada Letters, July-August 1983, pp. 25-35.

Taylor, Richard N. and Standish, Thomas A., "Steps to an Advanced Ada Programming Environment", Proc. 7th International Conference on Software Engineering, IEEE Computer Society, 1984, pp. 116-125.

3.1.2 PIC/Ada

The PIC/Ada language and toolset are one manifestation of the Precise Interface Control (PIC) project at the University of Massachusetts. PIC/Ada is a set of language features compatible with Ada, or Ada-like PDLs, intended to support the programming-in-the-large activities that are associated with preliminary design. Additional external information interfaces pertinent to PIC/Ada include module stubs, module interconnection descriptions and interface consistency reports. Module stubs represent the views of a given module that are appropriate for the purposes of one or more other modules. Module interconnection descriptions contain information about the access that a given module requests to other modules and the access to its own contents that the module is willing to grant to other modules. Interface consistency reports are the result of analyses of a preliminary design that can be performed by PIC tools. The internal information interfaces associated with PIC/Ada are internal representations similar to those employed by Arcturus (see section 4.1.1). These internal representations are themselves implemented as abstract data objects using a special purpose set of tools developed for constructing such representations. Thus the PIC/Ada toolset illustrates both information interfaces relevant to preliminary design and information interface technology.

A reference for PIC/Ada is:

Wolf, Alexander, Clarke, Lori and Wileden, Jack, "Ada-Based Support for Programming-in-the-Large", IEEE Software, Vol. 2, No. 2, March 1985.

3.2 Literature

The usual root information fragments found in the design activity of software development include the requirements specification as an input, and as outputs a series of design documents including one or more of a functional specification, architectural design, preliminary design, high-level design, interface design, database design, system design, subsystem designs, program designs and module designs. Some of these words turn out to be synonymous when used by certain people but not when used by others. Where a distinction is made between preliminary design and detailed design, the most common dividing line is that only the lowest level (usually called module or program design) is called detailed, with all others called preliminary (or some other term).

The non-root information fragments of design are less universal and depend upon the specific method and notation being used. Perhaps the most common fragments are some notion of component (component, unit, module, program) and a (usually hierarchical) structuring relationship among components. Others are inputs and outputs (to some level of detail), functions (either inherited from the requirements or derived from the functions of the requirements analysis) and data definitions used in interfaces and databases.

The following documents describe information interfaces related to the major design methods.

DeMarco, T., Structured Analysis and System Specification, Yourdon Press, 1978.

Fairley, R. E., "A Model of Software Structure", Proceedings of the Seventeenth Hawaii International Conference on System Sciences 1984, V. 1, 1984, 4-6 Jan. 1984.

Freeman, Peter; Wasserman, Anthony I., Tutorial on Software Design Techniques, (4th Edition), IEEE Computer Society Press, 1983.

Gilb, Tom, "Software Engineering Using Design By Objectives' Tools (DBO)", Software Engineering Notes, V. 9, No. 2, Apr. 1984, ACM SIGSOFT.

Gilb, Tom & Zvegintzov, Nicholas, Design by Objectives, North-Holland, 1984.

Orr, K.T., Structured Systems Development, Yourdon Press, 1977.

Page-Jones, Meilir, Practical Guide to Structured Systems Design, Yourdon Press, 1980.

Parnas, D.L., "On the Criteria to Be Used in Decomposing Systems into Modules," Communications of the ACM, Vol. 5, No. 2, Dec. 1972. pp. 1053-58. (Reprinted in Classics in Software Enginering, ed. E.N. Yourdon, Yourdon Press, 1979, pp. 141-50.)

Peters, Lawrence J., Structured Design: Methods and Techniques, Yourdon Press, 1981.

Warnier, J.D., Logical Construction of Programs, 3rd ed., trans. B. Flanagan, Van Nostrand Reinhold, 1976.

Weinberg, V., Structured Analysis, Yourdon Press, 1978.

Wirth, N., "Program Development by Stepwise Refinement," Communications of the ACM, Vol. 14, No. 4, April 1971, pp. 221-27.

Yourdon, E., and Constantine, L.L., Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design, 2nd ed., Yourdon Press, 1978.

The following documents describe information interfaces related to lesser known design methods, although not necessarily less important to the design of the JSSEE.

Braek, R., "Unified System Modelling and Implementation", Colloq Int de Commutation, Paris, Fr. May 7-11 1979, Publ by Comite du Colloq Int de Commutation, Paris, Fr. 1979, V.3, pp. 1180-1187.

Capella, G.; Di Leva, A.; Petrone, L.; Sirovich, F.; "Program Development and Documentation by Step-Wise Transformations: An Interactive Tool", Schneider, H. J. (Editor), Proceedings of the International Computing Symposium 1983 on Application Systems Development, Nurnberg, Germany 22-24 March 1983, Publ. B. G. Teubner, Stuttgart, Germany.

Cerchio, L.; Modesti, M.; Perardi, F.; Scrignaro, D., "A System Design Based on SDL Methodology", Cselt Rapp. Tec., V. 11, No. 6, Dec. 1983.

Estrin, G., "A Methodology for the Design of Digital Systems - Supported by SARA at the Age of One," Proceedings of the 1978 National Computer Conference, Vol. 47, AFIPS Press, 1978, pp. 313-32.

Futamura, Y., Kawai, T., Horikoshi, H., and Tsutsumi, M., "Development of Computer Programs by PAD (Problem Analysis Diagram)," Proceedings of the Fifth International Software Engineering Conference, IEEE Computer Society, 1981, pp. 325-32.

Hawryszkiewycz, Igor T., "A Semantic Design Method", IEEE Trans. on Software Engineering, V. SE-9, No. 4, July 1983, pp. 373-384.

Hoare, C.A.R, "Communicating Sequential Processes", Comm. ACM, V. 21, No. 8, August 1978.

Kanda, Yasunor; Sugimoto, Masakatsu, "Software Diagram Description: SDD and Its Application", Proc IEEE Comput Soc Int Comput Software Appl. Conf. 4th, COMPSAC 80, Chicago, IL, Oct. 27-31, 1980. Publ by IEEE, Piscataway, NJ 1980, pp. 300-305.

McDaniel, H., An Introduction to Decision Logic Tables, Petrocelli/Charter, 1978.

McQuillan, D., "Transaction Diagrams - A Design Tool," ACM SIGPLAN Notices, Vol. 10, No. 5, May 1975, pp. 21-26.

Morton, Richard and Freburger, Karl, "Toward Methodology for Functional Specification", Proc. COMPSAC 80, IEEE Computer Society Press, 1980.

Tichy, W. F., Software Development Control Based on Module Interconnection, Dept. of Computer Sci., Carnegie-Mellon Univ., Pittsburgh, PA, Proceedings of the 4th International Conference on Software Engineering, Munich, Germany, 17-19 Sept. 1979.

The following documents are also related to traditional design approaches, but relate specifically to preliminary design.

Babb, R. G., II, "Data-Driven Implementation of Data Flow Diagrams", Dept. of Computer Sci. and Engng., Oregon Graduate Center, Beaverton, OR, Sixth International Conference on Software Engineering, Tokyo, Japan, 13-16 Sept. 1982.

Dreyfus, J. M.; Karacsony, P. J., "The Preliminary Design as a Key to Successful Software Development", 2nd International Conference on Software Engineering, San Francisco, CA, 13-15 Oct. 1976.

Goldman, Neil M., Three Dimensions of Design Development, ISI/RS-83-2, University of Southern California, Information Sciences Institute, July 1983.

Huff, S. L., "Preliminary Design for Complex Software Systems Using Graph Decomposition", Proceedings of the International Conference on Cybernetics and Society, Boston, MA, 8-10 Oct. 1980, pp. 479-484.

Huff, S. L., "A Methodology for Supporting System Architects During Preliminary Design", Inf. and Manage., V. 5, No. 4-5, Sept.-Nov. 1982, pp. 259-268.

Matsumoto, Yoshihiro, "Software Design Methodology: Bridge From Requirements Specification to Software Design", Jpn Annu Rev Electron Comput Telecommun Comput Sci Techno 1982. Publ by Ohmsha Ltd., Tokyo, Japan and North Holland Publishing Co., Amsterdam, Netherlands 1982, pp. 175-192.

In addition to the root fragments found commonly, others that are less common are the outputs of various design analyses. These include traceability of requirements, completeness reports, consistency checks and design review reports. The following documents describe information interfaces of this type.

Beane, J.; Giddings, N.; Silverman, J., "Quantifying Software Designs", Proceedings of the 7th International Conference on Software Engineering, Orlando, FL, 26-29 March 1984, pp. 314-322.

Lamb, S.S.; Leck, V.G.; Peters, L.J. and Smith, G.L., "SAMM: A Modeling Tool for Requirements and Design Specification", Proceedings of COMPSAC 78, IEEE Computer Society, 1978, pp. 101-08

Leveson, Nancy G.; Stolzy, Janice L., Analyzing Safety and Fault Tolerance Using Time Petri Nets, Technnical Report No. 220, University of California, Irvine, Dept. of Information and Computer Science, July 1984.

Romanos, J. P., "The Software Design Processor", Proceedings of COMPSAC the IEEE Computer Society's Third International Computer Software and Applications Conference, Chicago, IL 6-8 Nov. 1979.

One area of considerable disagreement related to the information interfaces of design is the temporal relationships among the design products. Two areas where this is particularly true are decomposition versus composition and the control-first versus data-first approaches. The following documents describe design information interfaces that relate to design approaches that are less common, but still not considered futuristic.

Diaz-Herrera, Jorge L., "Pragmatic Problems with Step-Wise Refinement Program Development", Software Eng. Notes, V. 9, No. 2, Apr. 1984, ACM SIGSOFT.

Gibbs, Simon, Tsichritzis, Dionysis, "A Data Modeling Approach for Office Information Systems", ACM Trans. on Office Information Systems, V. 1, No. 4, October 1983, pp. 299-319.

Jackson, M.A., Principles of Program Design, Academic Press, 1975.

Jefferson, David K., Information System Design Methodology: Global Logical Data Base Design, David W. Taylor Naval Ship Research and Development Center Report DTNSRDC-82/057, August 1982.

Orman, Levent, "An Array Theoretic Specification Environment for the Design of Decisions Support Systems", International J. of Policy Analysis and Information Systems, Vol. 6, No. 4, 1982, pp. 373-391.

The following documents discuss formal approaches to design specifications or special design languages, but do that within the usual context of design.

Bauer, F.L., et al., "Programming in a Wide Spectrum Language: A Collection of Examples", Science of Computer Programming, Vol. 1, No. 1, October 1981.

Bjorner, Dines, and Jones, Cliff B., Formal Specification and Software Development, Prentice-Hall International, 1982.

Black, Andrew P., Report on the Programming Notation 3R, (Technical Mono), Oxford Univ. (England), Programming Research Group, Report No. PRG-17, 1980.

Clemmensen, G. B.; Oest, O. N.," Formal Specification and Development of an Ada Compiler-A VDM Case Study", Proceedings of the 7th International Conference on Software Engineering, Orlando, FL, 26-29 March 1984, pp. 430-440.

De Santo, R.J., "Using Finite-State and Structured Design Techniques for Embedded Software Design:, Proceedings of the IEEE 1978 National Aerospace and Electronics Conference, NAECON, 16-18 May, 1978, Dayton, OH, pp. 236-241.

Luckman, David, and von Henke, Fredrich W., "An Overview of Anna, a Specification Language for Ada", IEEE Software, Vol. 2, No. 2, March 1985, pp. 9-22.

Markowitz, H.M., Malhotra, A., Pazel, D.P., "The EAS-E Application Development System: Principles and Language Summary", Comm. ACM, Vol. 27, No.8, August 1984, pp. 785-799.

Mekly, Leon J.; Yau, Stephen S.; "Software Design Representation Using Abstract Process Networks", IEEE Trans Software Eng., V.SE-6, Sept. 1980, pp. 420-435.

Schneider, Hans-Jochen, ed., "Formal Models and Practical Tools for Information System Design", Proceedings of the IFIP TC-8 Working Conference on Formal Models and Practical Tools for Information System Design, North-Holland, 1979.

Sufrin, Bernard, Formal Specification of a Display Editor,

(Technical Mono.), Oxford Univ. (England), Programming Research Group, Report No. PRG-21, 1981

An important part of preliminary design is database design. References related to database design can be found in Section 18.1 below.

When detailed design is a separate phase, the product of that activity is usually the design of the algorithm of a single component. The form of that design depends upon the notation used, such as flowcharts or pseudocode. In either case, however, the non-root fragments are almost always those of programming control structures, such as selection, iteration and procedure call. Hence, the information fragments of detailed design are largely independent of notation, except that some notations allow more control structures than others (cf. Nassi-Shneiderman diagrams versus flowcharts). The following documents describe information interfaces that are normally associated with detailed design.

Belcastro, Richard J., "Specification Template Speeds Software Design", EDN, V.27, Oct. 27, 1982, pp. 233-236.

Caine, S.H., and Gordon, E.K., "PDL - A Tool for Software Design," Proceedings of the 1975 National Computer Conference, Vol. 44, AFIPS Press, 1975, pp. 271-76.

Callender, E.; Hartsough, C.; Kleine, H., "SDDL: Software Design and Documentation Language", Jet Propulsion Lab., Pasadena, CA, Proceedings of the NBS/IEEE/ACM Software Tool Fair, San Diego, CA, 10-12 March 1981.

Chapin, N., "New Format for Flowcharts," Software - Practice and Experience, Vol..4, No. 4, Oct.-Dec. 1974, pp. 341-57.

Guttag, J. V.; Horning, J. J., Preliminary Report on the Larch Shared Language, CSL-83-6, Xerox Palo Alto Research Centers, December 1983.

Katzan, H., Jr., Systems Design and Documentation: An Introduction to the HIPO Method, Van Nostrand Reinhold, 1976.

Lucas, Peter; Thatcher, James W.; Zilles, Stephen N., A Look at Algebraic Specifications, IBM Research, n.d.

Nassi, I. and Shneiderman, B., "Flowchart Techniques for Structured Programming," ACM SIGPLAN Notices, Vol. 8, No. 8, August 1973, pp. 12-26.

Mohri, T.; Ono, E.; Sato, H.; Uehara, S., "PDAS: An Assistant for Detailed Design and Implementation of Programs", Proceedings of the 7th International Conference on Software Engineering, Orlando, FL, 26-29 March 1984.

One of the goals of the STARS project is to promote reusability. This subject pertains to both design and code, and much of the current research in this area does not distinguish between them. Reusability is also considered by some to be a subset of rapid prototyping, which we have listed under future paradigms below. The reader should, therefore, look at the literature on rapid prototyping to find out more about reusability. There is also much research into Ada-specific reuse of packages; the reader is encouraged to look in this area as well.

Biggerstaff, T., Editor, Proc. ITT Workshop on Reusability in Programming, ITT Programming, 1983. (Selected papers reprinted in IEEE Transactions on Software Engineering, V. SE-10,No. 5, Sept. 1984.)

Bowen, Thomas P.; Post, Jonathan V.; Presson, P. Edward; Schmidt, Robert; Tsai, Juitien, Software Interoperability and Reusability,V. 1-Final Report, Rome Air Development Center, Griffiss AFB, NY, July 1983.

Bowen, Thomas P.; Post, Jonathan V.; Presson, P. Edward; Schmidt, Robert; Tsai, Juitien, Software Interoperability and Reusability, V.2 - Guidebook for Software Quality Measurement, Rome Air Development Center, Griffiss AFB, NY, July 1983.

C.V. Ramamoorthy, Ed., IEEE Transactions on Software Engineering, Vol. SE-10, No. 5, September 1984, Special Issue on Software Reusability.

The following document does not fit easily into one of the previous groupings of design information interfaces. It is interesting, however, because it identifies some information fragments that, if generated during design, would greatly simplify the process of database conversions whenever that becomes necessary.

Gallagher, Leonard J., "Database Conversions Demand Common Standards for Data Structures", Data Management, January 1985.

4.0 Code, Unit Test and Debug

4.1 Tools Review

4.1.1 Arcturus

The Arcturus system [Standish, 1983], [Standish and Taylor, 1984], developed at the University of California at Irvine, provides a variety of interesting information interfaces.

Arcturus was an experiment in supporting highly interactive creation and debugging of Ada programs. Its style and user interface were modelled on the InterLisp environment. Among its information interfaces is an interactive Ada program design language (PDL), which is described in section 3.1.1. Here we describe those information interfaces of Arcturus pertinent to coding (interactive Ada and the Arcturus internal representation) and debugging (interactive Ada and performance measurement output).

Interactive Ada is in fact a subset of Ada, supported by tools to permit interactive execution at a statement-by-statement level. The subset is essentially the Pascal-superset part of Ada -- the basic sequential control constructs and statements, plus packages. It omits tasking, defers type-checking to run-time and does not permit overloading. Interactive Ada can be processed by the Arcturus syntax-checker, pretty printer and template-driven editor. It is primarily interesting, however, because of the Arcturus tools that permit individual expressions or statements in interactive Ada to be interpreted as soon as they are entered by a user. This feature provides users with immediate feedback on a program as it is being developed. It also permits the interactive Ada language itself to serve as a useful debugging tool. To query the current value of any variable in an interactive Ada program, for example, the user merely types that variable's name. The interactive Ada interpreter evaluates the expression consisting of that name and prints the result, which is just the variable's current value. Thus interactive Ada is representative not only of the standard information interface which is a programming language, but also of an information interface facilitating program debugging.

Another Arcturus information interface relevant to coding, although it is also relevant to design, is the Arcturus internal representation. Unlike the other Arcturus information interfaces considered here and in section 3.1.1, all of which are external interfaces, this one is an internal interface. The internal representation is constructed from an Arcturus PDL or interactive Ada program and used as a basis for interpretation. It can also be translated into a form suitable for compilation by an Arcturus tool known as the "export laundry", while compiled forms of Ada programs can be translated into the internal representation by a complementary "import laundry" tool. Of course, the compiled version of an (interactive or otherwise) Ada program is itself an information interface. The Arcturus internal representation consists of nested records, linked into a tree data structure and annotated with symbol tables. In essence, the tree of nested records is an abstract syntax tree which constitutes a condensed representation of a program's semantics. This representation provides a convenient information interface for tools such as the interpreter or various proposed analysis tools that must operate on the semantic aspects of the program.

A final class of information interfaces found in Arcturus is

performance measurement output. This external information interface is particularly useful during debugging. In Arcturus, performance measurement output can be provided in the form of histograms indicating what percentage of execution time is spent in various parts of a program. Alternatively, a color-coded version of the program can be presented, with "hotter" colors (toward red) indicating those parts of the program whose execution consumes the most time while "cooler" colors (toward blue) indicate those parts consuming the least time. In either manifestation, this feature of Arcturus represents another interesting, and relatively unusual, type of information interface.

References for Arcturus are:

Standish, Thomas A., "Interactive Ada in the Arcturus Environment", Ada Letters, July-August 1983, pp. 25-35.

Taylor, Richard N. and Standish, Thomas A., "Steps to an Advanced Ada Programming Environment", Proc. 7th International Conference on Software Engineering, IEEE Computer Society, 1984, pp. 116-125.

4.1.2 Toolpack

The Toolpack environment [Osterweil, 1983], created by a consortium of university, government and industrial research laboratories, is an experimental environment created to support the development of numerical analysis software in Fortran. It contains several representative information interfaces, primarily related to coding, testing and debugging. Perhaps of greater interest is Toolpack's information interface technology, which is contained in the Odin subsystem. Here we describe the Toolpack information interfaces, while Odin's information interface technology is discussed in section 4.1.3.

One class of information interfaces found in Toolpack are those related to the coding, compilation and execution of programs. The main external information interfaces are program units and program unit groups (PUGs). The program unit is simply a Fortran program unit, while a PUG is a named collection of program units. Various versions of program units and PUGs may exist among the information interfaces of a Toolpack system, including Ratfor versions, various dialects of the Fortran code, and versions conforming to various text formats. Internal information interfaces in this class include token streams, parse trees, symbol tables and object modules. Various versions of parse trees, resulting from program transformations (such as to change dialects of Fortran) might be among these internal information interfaces.

Another class of information interfaces found in Toolpack are those related to analysis, testing and debugging. The external information interfaces in this class are test data collections

(TDCs) and error reports. TDCs consist of test data sets and optional output specifications. Error reports list semantic errors discovered through static analysis, data flow analysis, or portability checking. Internal information interfaces in this class include annotated flow graphs and versions of programs instrumented to support debugging.

A third class of information interfaces found in Toolpack are those related to invocation of Toolpack tools. These external information interfaces include options packets, which provide customizing directives to tools for specific applications, and procedures, which are predefined sequences of Toolpack commands. Although our interests in this report do not extend to the invocation interfaces of environments, we mention these invocation-related information interfaces here in the interest of completeness.

It should be noted that, although information interfaces can be thought of as the contents of a project database, Toolpack explicitly separates this conceptual viewpoint from the actual implementation of its information interfaces. The result is that only a small number out of the many information interfaces listed above may actually be physically present in a Toolpack database at any given time, but that from the user's point of view they may all be considered to be present there. This situation arises from the approach to defining information interfaces employed in Toolpack, which is discussed in the section on Odin, 4.1.3 below.

A reference for Toolpack is:

Osterweil, Leon J. "Toolpack -- An Experimental Software Development Environment Research Project", IEEE Transactions on Software Engineering, November 1983, pp. 673-685.

4.1.3 Odin

The Odin system [Clemm, 1984] is intended to serve as the basis for an extensible program development environment. Odin itself is an outgrowth of the Toolpack project and an extension of the Integrated System of Tools (IST) subsystem found in Toolpack [Osterweil, 1983]. For our purposes the main capability provided by Odin is for users to define information interfaces and their relationships to other information interfaces and to tools in a software development environment.

Odin's approach is to treat all information interfaces as files or sets of files and to provide mechanisms allowing users to define file types and operations specified in terms of those file types. The various file types may be thought of as distinct views of some part of a software system. For example, test.f:fmt and test.f:obj might be two file types corresponding to two different views of the Fortran program stored in the file "test" (the extension .f denotes a Fortran base (primitive) object type in Odin), the former a formatted view of the source and the latter an

object code view. As this example indicates, Odin supports the definition of both external and internal information interfaces.

Two languages are provided by Odin. One is a command language, which permits a user to request the creation of a view. Essentially, this language lets a user name a view (file type or version of an information interface) that is desired (e.g., test.f:obj) and optionally indicate where a copy of that view should be stored (e.g., test.f:obj›test2.f:obj). The other Odin language is a specification language in which users can define new views (file types or information interfaces) and indicate how those views are to be created. Odin transforms this information into a derivation graph, which indicates what tools can be used to create a file (information interface) of one type from a file of some other type. This graph then encodes the relationships among information interfaces and tools, and provides a kind of representation of all the possible information interfaces in a given environment as well as the kinds of operations that can be performed on them. Using this representation, Odin can decide which types of views (information interfaces) should be physically stored in its database and which can be generated on demand, through the application of one or a sequence of tools. Information interfaces in the latter group may then be physically stored or not, depending on other considerations such as file system capacity, but are still conceptually present from the user's viewpoint.

A reference for Odin is:

Clemm, Geoffrey M. "ODIN - An Extensible Software Environment; Report and User's Reference Manual", Technical Report CU-CS-262-84, Department of Computer Science, University of Colorado, 1984.

4.1.4 Diana and IDL

Diana, a Descriptive Intermediate Attributed Notation for Ada, was developed by researchers from Carnegie-Mellon University, the University of Karlsruhe and Tartan Laboratories. It is based on two earlier proposals for intermediate forms for Ada -- TCOL (or more precisely, the Ada version of TCOL), developed by the PQCC project at Carnegie-Mellon, and AIDA, developed at the University of Karlsruhe.

Diana is intended as an intermediate representation for Ada programs. As such, it represents a natural information interface between the front end and back end of a compiler. It is also intended, however, to be useful as an information interface between other tools in an Ada software development environment. Diana is designed to encode the information about an Ada program that can be derived from lexical, syntactic and static semantic analyses, but not to contain information resulting from dynamic semantic analysis, optimization or code generation.

Diana is most properly viewed as an abstract data type, defining a class of information interfaces. Any instance of that abstract data type, i.e., any specific information interface, is a Diana representation of a specific Ada program. Diana itself defines a set of operations providing the only means for accessing or modifying an instance of the abstract data type. Although the Diana reference manual offers example implementations for the abstract data type, these only constitute constructive proofs that Diana can be implemented and do not serve to define what Diana actually is.

The concept of an attributed tree serves as a conceptual model for Diana. That is, a Diana representation of an Ada program may be thought of (though it need not be implemented as) a tree of nodes, each of which may have a set of attributes associated with it. Therefore, the definition of Diana is given in terms of a set of classes of nodes and the attributes associated with each class. Conceptually, at least, all information about an Ada program's syntax and static semantics is captured in this attributed tree. Hence, unlike the Arcturus internal form, Diana does not include a conceptually distinct symbol table, but rather incorporates all the information traditionally contained in a symbol table in the attributed tree. An implementation of this conceptual attributed tree may, of course, actually employ symbol tables if the implementors so choose.

The Interface Description Language, IDL, was developed at Carnegie- Mellon and employed by the developers of Diana. IDL provides a notation in which abstract descriptions of a class of data objects, especially information interfaces, can be formulated.

IDL is especially well-suited for describing information interfaces since it treats all data objects as (possibly degenerate) attributed trees. The main descriptive capabilites of IDL are aimed at defining the various classes of nodes and their associated attributes comprising such a structure. In fact, the Diana reference manual uses IDL to present the definition of Diana.

A primary motivation for IDL was the need to facilitate interchange of information among tools in a software development environment that might want or need to use different internal representations for the same information. Therefore, IDL concentrates on supporting abstract descriptions of information interfaces rather than description of the implementation details related to those interfaces. IDL does, however, permit specification of some implementation details through a 'representation specification' feature similar to Ada's.

Associated with IDL is a processor for translating IDL definitions of information interfaces into implementations that can be used by the tools in a software development environment.

This processor assumes a somewhat restrictive structure for tools and their interactions, but given that structure it is sufficient to automate the process of going from an IDL description to the data structure definitions and code necessary to implement the interface defined by that description.

References for Diana and IDL are:

Evans, A. Jr. and Butler, K. J. (eds.), "Diana Reference Manual(Revision 3)", Technical Report TL 83-4, Tartan Laboratories Inc.,Pittsburgh, Pennsylvania, February 1983.

Lamb, D. A., "Sharing Intermediate Representations: The Interface Description Language", Technical Report CMU-CS-83-129, Computer Science Department, Carnegie-Mellon University, Pittsburgh,Pennsylvania, May 1983.

Nestor, J. R., Wulf, W. A. and Lamb, D. A., "IDL - Interface Description Language: Formal Description", Technical Report,Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania, February 1983.

4.1.5 EDL High-Level Debugging Toolset

The Event Definition Language (EDL) is part of an approach to high-level debugging developed at the University of Massachusetts. This approach, called Behavioral Abstraction, is an alternative to traditional debugging techniques, which are based on a detail-intensive, unit-at-a-time perspective. In the Behavioral Abstraction approach a system's activity is viewed as consisting of a stream of significant, distinguishable behaviors, termed 'events'. Selective clustering of sequences of events into higher level events produces increasingly abstract views of the system. This approach permits a user to deal with only the level of detail that is appropriate for the current debugging problem, rather than being perpetually mired in low level details of system activity.

The Event Definition Language has been created to facilitate the description of behavioral patterns that a system might exhibit. Users of EDL use primitive and previously defined events, together with a set of event sequencing operators and a means for filtering events based on their characteristics, to describe behavioral patterns of interest in the system. Thus software developers can use EDL to flexibly define viewpoints that highlight those aspects of system behavior that are relevant to specific questions currently under investigation.

A prototype set of debugging tools supporting the Behavioral Abstraction framework and the Event Definition Language implements some of the fundamental capabilities required for high-level debugging. Specifically, it permits users to monitor behavior of a system from any arbitrary high-level perspective that they may wish to define.

The EDL definitions of viewpoints for use in debugging a system represent another kind of information interface. These definitions are both internal and external, since they are created and used by the developers who wish to debug the system and they are also used internally by the tools composing the Behavioral Abstraction debugging toolset.

References for EDL are:

Bates, P. and Wileden, J. "High Level Debugging of DistributedSystems", Journal of Systems and Software, 3, 4, (December 1983),pp. 255-264.

Bates, P. and Wileden, J. "An Approach to High-Level Debugging of Distributed Systems", Proc. SIGPLAN/SIGSOFT Symposium onHigh-Level Debugging, Asilomar, California (March 1983), pp. 24-33.

4.2 Literature

The code, unit test and debug activity uses information interfaces related to each of coding, testing and debugging. The subject of testing is dealt with in Sections 5 and 9, as well as here.

The obvious root information fragment of coding is the source code of the unit being coded. Others include the results of various static and dynamic analyses that may be performed, as well as the outputs of the compilers, interpreters, linkers and loaders that provide the transition between coding and testing. The non-root fragments of coding depend somewhat on the syntax and semantics of the programming language being used, particularly where special purpose languages are used, and on the nature of the language translation systems used (compiler, linker and loader, or whatever). The following documents describe information interfaces related to coding.

Arnon, J.; Lehrhaupt, H., "Software Documentation an Automated Approach", Proceedings of Trends and Applications 1982. Advances in Information Technology, Gaithersburg, MD 27 May 1982, pp. 57-64.

Diaz-Herrera, J.L., Flude, R.C., "PASCAL/HSD: A Graphical Programming System", Proc. COMPSAC 80, IEEE Computer Society Press, 1980.

Evans, A., Jr., and Butler, K.J., eds., Diana Reference Manual (Revision 3), Report TL 83-4, Tartan Laboratories, Inc., February, 1983.

Gries, David, The Science of Programming, Springer-Verlag, 1981.

Kojima, K.; Nakajima, R; Yuasa, T., "The Iota Programming

System-A Support System for Hierarchical and Modular Programming", Lavingston, S. (Editor), Information Processing '80. Proceedings of the IFIP Congress 80, Tokyo, Japan, 6-9 Oct. 1980, Publ. North-Holland, Amsterdan, Netherlands, pp. 299-304.

Linger, R.C.; Mills, H.D.; Witt, B.I., Structured Programming: Theory and Practice, Addison-Wesley Pub. Co., 1979.

Millard, D. P., "Automated Documentation for Real-Time Software", IEEE Southeastcon '83 Conference Proceedings, Orlando, FL 11-14 April 1983, pp. 78-80.

The root information fragments of unit testing include special purpose code (stubs and drivers), and test plans, specifications, procedures, data and results. Stubs and drivers are simply instances of more code. The non-root fragments of plans, specifications and procedures are usually defined by standards, if any, covering these topics. Test data generation and the kinds of results which should come from testing are the subjects of several of the references below. The following documents describe information interfaces related to unit testing.

Bourgonjon, R. H., "Chill Testing", IEEE Proceedings COMPSAC 83: The IEEE Computer Society's Seventh International Computer Software and Applications Conference, Chicago, IL, 7-11 Nov. 1983, pp. 245-246.

DeMillo, Richard A., Program Mutation: An Approach to Software Testing, U.S. Army Research Office, Triangle Park, NC, April 1983.

Gannon, John D.; McMullin, Paul R., "Combining Testing with Formal Specifications: A Case Study", IEEE Trans. Software Eng., V. SE-9 3 May 1983, pp. 328-335.

Debugging, in the sense implied by coupling it with coding and unit testing, is usually an informal activity in which the programmer interacts with the program to diagnose and correct the defects detected during unit testing. The fragments of debugging depend highly on the debugging tools available and being used. Most interactive debuggers, whether source language or machine language oriented, include such concepts as breakpoints, instructions or statements, variables, values and locations of code and data. Debuggers also depend on symbol tables and load maps. Batch debugging environments include dumps, traces and snap shots. In more formally controlled situations there may be defect analysis reports and corrective action reports that are generated.

The following documents describe information interfaces related to debugging.

Casey, Dan; Fellows, Jon; Finfer, Marcia, Software Debugging Methodology. Volume III. Literature and Site Surveys, Rome Air Development Center, Griffiss AFB, NY, Report No. RADC-TR-79-57-Vol.-3, April 1979.

Dean, A.; Gaines, J.; McCoy, W., "HOL Debug and Test in the Trident Fire Control Environment (Weapons Systems)", Kirk, D. E. (Editor), IEEE Sixteenth Asilomar Conference on Circuits, Systems and Computers, Pacific Grove, CA, 8-10 Nov. 1982, 1983, pp. 162-166.

Dunn, R., Software Defect Removal, McGraw-Hill, 1984.

Weiser, Mark, "Program Slicing", IEEE Trans. on Software Engineering, V. SE-10, No. 4, July 1984, pp. 352-357.

Smith, Truck, Secrets of Software Debugging, TAB Books, 1984.

5.0 Integration and System and Acceptance Test

See also validation and verification.

Beizer, B., Software System Testing and Quality Assurance, Van Nostrand Reinhold, 1984.

Beizer, Boris, Software Testing Techniques, Van Nostrand Reindhold, 1982.

Compton, M. T., "Testing with Inspection Procedures (Computer Software)", IEEE Conference Digest of the International Electrical, Electronics Conference and Exposition, Toronto, Canada, 2-4 Oct. 1979, pp. 22-23.

Greenspan, A. M., "Real Solution for Enhancing Productivity in TPS Development", AUTOTESTCON '83. 'New Horizons in Automatic Testing', Fort Worth, TX, 1-3 Nov. 1983, pp. 122-127.

Haley, Allen; Zweben, Stuart, An Approach to Reliable Integration Testing (Technical Report), Computer and Information Science Research Center,Air Force Office of Scientific Research, Bolling AFB, DC, Report No. OSU-CISRC-TR-81-5; AFOSR-TR-81-0578, May 1981.

Hetzel, W.C., ed., Program Test Methods, Prentice-Hall, 1973.

Hicks, T. C., "System-Level Software Testing", Electron Test, V. 6, No. 9, Sept. 1983, pp. 61-3.

Janjsz, Paul E.; Turoczy, William R., Application of Software Test Tools to Battlefield Automated Systems, U.S. Army

Armament Research and Development Center, Dover, NJ, July 1984.

Klinger, D. R., "Integrating Real-Time Software with a System Tester", Kirk, D. E. (Editor), IEEE Sixteenth ASILOMAR Conference on Circuits, Systems and Computers, Pacific Grove, CA, 8-10 Nov. 1982, pp. 535-538.

Lozier, D. W.; Maximon, L. C.; Sadowski, W. L., "Performance Testing of a Fortran Library of Mathematical Function Routines - a Case Study in the Application of Testing Techniques", Journal of Research of National Bureau of Standards Sect. B, Math Sci., V. 77B, 3-4 Jul-Dec 1973, pp. 101-110.

Miller, Edward and William E. Howden, Tutorial: Software Testing and Techniques", IEEE Computer Society, 1981.

Perry, William E., A Structured Approach to Systems Testing, Prentice-Hall, 1983.

Spencer, Richard H., Planning, Implementation and Control in Product Test and Assurance, Prentice-Hall, 1983.

6.0 Deployment, Maintenance and Support

6.1 Tools Review

6.1.1 GTE Change Tracking System

The GTE Change Tracking System (CTS) [Vanderlei, 1983] provides a typical example of tools used in maintenance and support. This tool implements a system for managing problem-handling, from initial report to final disposition.

The CTS introduces a set of related information interfaces into the GTE software development environment. One of these is the CTS problem report and tracking form, which is originated when a problem is reported and then is updated at each step in resolving that problem. Another information interface related to CTS is a statistical summary of CTS activities, i.e., number of reported problems, current status of those problems, etc.

A reference for CTS is:

Vanderlei, Kenneth W., "Software Development Methodology and Practices", GTE Network Systems Journal, Third Quarter 1983, pp. 76-82.

6.2 Literature

See also configuration management.

Bernstein, I; Yubas, C. M., "Software Manufacturing", Des Autom Conf. 15th Proc., Las Vegas, NV 19-21 June 1978, pp. 455-462.

Dorr, Oscar J., "Software Logistics", Logistics Spectrum, Journal of the Society of Logistics Engineers, Vol. 18, No. 1, Mar. 1984.

Francis, Webster E. Maj., Distribution of Software Changes for Battlefield Computer Systems: A Lingering Problem, U.S. Army Training and Doctrine Command, Ft. Monroe, VA, Jun. 1983

Glass, Robert L.; Ronald A. Noiseux, Software Maintenance Guidebook, Prentice-Hall, 1981.

Hall, John F. LT III, Documentation for Software Maintenance, Naval Postgraduate School, Monterey, CA, Dec. 1983.

Herndon, M.A., and McCall, J.A., "A Tool for Software Maintenance Management", Proc. A Conference on Software Development Tools, Techniques, and Alternatives, July 25-28, 1983, Arlington, Va., IEEE Computer Society Press.

Ince, D. C., "The Provision of Procedural and Functional Interfaces for the Maintenance of Program Design Language and Program Language Notations", Sigplan Not., V. 2, Feb. 1984, pp. 68-74.

Laprie, J. C., "Evaluating the Dependability of Operational Software", Tech. and Sci. Inf. (France), V.2, No. 4, 1983, pp. 221-234.

Marca, D., "Software Manufacturing and Large Software Maintenance", Digest of Papers COMPCON Spring '84, Twenty-eighth IEEE Computer Society International Conference, San Francisco, CA, 27 Feb-1 Mar 1984, pp. 312-315.

Parikh, Girish, Techniques of Program and System Maintenance, Winthrop Publishers, Inc., 1982.

Parikh, G. and N. Zvegentzov, Tutorial on Software Maintenance, IEEE, 1983.

Schneidenwind, Norman F., Software Maintenance: Improvement through Better Development Standards and Documentation, Naval Postgraduate School, Monterey, CA, Report No. NPS-54-82-002, 22 Feb. 82.

Upchurch, Robert B. LT, Improvements to Software Maintenance Methods in Real Time Embedded Aviation Flight Systems, Naval

Postgraduate School, Monterey, CA, Dec. 1983.

7.0 Project Management

7.1 Tools Review

7.1.1 Enhanced Project Evaluation Schedule Tool

The Enhanced Project Evaluation Schedule Tool (EPEST) [Rice, 1983] is a typical example of a project management tool. This tool offers cost estimation, task scheduling and progress monitoring support. The information interfaces required for EPEST include cost estimation charts, weekly group progress reports, milestone charts, and activity schedules.

A reference for EPEST is:

Rice, Verner, "Software Development Tools: Goals and Status", GTE Network Systems Journal, Third Quarter 1983, pp. 83-86.

7.2 Literature

The information interfaces of project management are related to planning and controlling software projects. At the root level, the fragments include project plans, status reports, and budgets. At the non-root level, these include a variety of alternative ways of recording each of the root fragments. For example, project plans can include PERT diagrams, which, in turn, are made up of activities, durations and dependency relations. In general, project managers deal with tasks, resources and relationships among them.

Abdel-Hamid, T. K.; Madnick, S. E., "A Model of Software Project Management Dynamics", Proceedings of COMPSAC 82, IEEE Computer Society's Sixth International Computer Software and Applications Conference, Chicago, IL, 8-12 Nov. 1982, pp. 539-554.

Agresti, W; Card, D.; Church, V.; McGarry, F., Managers Handbook for Software Development, National Aeronautics and Space Administration, Greenbelt, MD, Report No. NAS 1.15:85604; SEL-84-001; NASA-TM-85604, Apr 1984.

Bernstein, L., "Software Project Management Audits", J. Syst. and Software, V. 2, No. 4, Dec. 1981, pp. 281-287.

Boehm, Barry W., Software Engineering Economics, Prentice-Hall, 1981.

Campbell, I.; Stilling, C., "GALAAD: A Software Management System", Productivity and Data Processing: Two Essentials for a Dynamic Company. Proceedings of the Spring Convention, V. 2, Paris, France, 30 May-3 Jun 1983, pp. 208-212

Daly, Edmund B.; Minichowicz, Donald A., "Management of Large Software Development for Stored Program Switching Systems", GTE Autom Electr., J.V. 17, 5 Sept. 1979, 155-160.

DeGiorgio, A.; Esposito, F.; Ingravallo, G.; Mascolo, R., "A System of Documentation Standards for Software Project Management", AICA '79 Conference, Bari, Italy, 10-13 Oct. 1979, pp. 167-172.

Evans, Michael W., Productive Software Test Management, Wiley-Interscience, 1984.

Evans, Michael W.; Piazza, Pamela; Dolkas, James B., Principles of Productive Software Management, Wiley and Sons, 1983.

Ferrentino, A. B., "Software Manager's Workstation", EASCON '83: 16th Annual IEEE Electronics and Aerospace Systems Conference, Washington, DC, 19-21 Sept. 1983.

Forshee, Michael et al., Handbook for Evaluation and Life Cycle Planning for Software, V. 2 - Contract Management, Electronic Systems Division, Air Force Systems Command, Hanscom AFB, MA, Feb. 1983.

Forshee, Michael et al., Handbook for Evaluation and Life Cycle Planning for Software, V. 3 - Reviews Audits and CPCI Specifications, Electronic Systems Division, Air Force Systems Command, Hanscom AFB, MA, Feb. 1983.

Freeman, Peter, Wasserman, Anthony I., Houghton, Raymond C., Jr., "Comparing Software Development Methodologies for Ada: A Study Plan", Software Engineering Notes, V. 9, No. 4, July 1984, ACM SIGSOFT.

Matsumoto, Yoshihiro, "Management of Industrial Control Software Production", Computer Magazine, V. 17, No. 2, February 1984, pp. 59-72.

Metzger, P.W., Managing a Programming Project, Prentice-Hall, 1973.

Moorehead, Donald F. LCDR; Ransbotham, James T. Jr. LCDR, A Program Manager's Methodology for Developing Structured Design in Embedded Weapon Systems, Naval Postgraduate School, Monterey, CA, Dec. 1983.

Neumann, Albrecht J., Management Guide for Software Documentation, NBS, Systems & Software Technology Div., Washington, DC, National Bureau of Standards Spec. Publ. 500-87, Jan. 1982.

Patrick, Robert L.; Ware, Willis H., Perspectives on

Oversight Management of Software Development Projects, The Rand Corp., Santa, Monica, CA, July 1983.

Tausworthe, R. C., "The Work Breakdown Structure in Software Project Management", J. Syst. and Software, V. 1, No. 3, 1980, pp. 181-186.

Walker, M.G., Managing Software Reliability: The Paradigmatic Approach, Elsevier/North-Holland, 1980.

Wolberg, John R., "Costing Model for Software Conversions", Software Pract. Exper, V. 12, 11 Nov. 1982, pp. 1043-1049.

8.0 Configuration Management and Version Control

8.1 Tools Review

8.1.1. GTD-5 EAX Software Management Support System

The GTD-5 EAX Software Management Support System (SMSS) [Chauza and Fortune, 1981] is a typical facility for configuration management. It consists of two major components, the UNIX/PWB Source Code Control System (SCCS) and a Load Generation System (LGS). These tools require the introduction of several additional information interfaces into the GTD-5 software development environment.

One class of information interfaces needed by SMSS controls and records access to other information interfaces in the environment. Access control records are internal information interfaces that describe the access privileges that various users of the environment have with respect to various other information interfaces (such as source code for specific system modules). A transaction log is another internal information interface in SMSS, used to record all uses and/or modifications of other information interfaces (such as source code modules).

Another class of information interfaces needed by SMSS represents changes made to other information interfaces. For example, when the source code for some module is modified, SMSS stores a record of the modifications (called a delta) that were made along with the original version of the source code module. Keeping this historical trace rather than updated versions of the module permits SMSS to reproduce any version of the module that might be requested without the necessity of storing multiple complete copies of that module.

The LGS mainly depends upon previously mentioned information interfaces, such as source code modules and deltas. It produces various other information interfaces, including listings and error reports, object modules, and load modules.

A reference for SMSS is:

Chauza, Edward J. and Fortune, Larry E., "GTD-5 EAX Software Management Support System", GTE Automatic Electric Journal, May-June 1981, pp. 90-96.

8.2 Literature

The root information fragments of configuration management include documents and programs (modules, subsystems, systems), each associated with baselines, revisions and versions. Configuration management also deals with problem reports, change authorizations and correction reports.

The non-root information fragments of configuration management are the components of each of these. Documents may be made up of words, lines, paragraphs or chapters, depending upon the nature of the system used to support changes to them. Programs are frequently changed by updating lines. Change control documents are usually made up of fields of information.

There are a number of companies that have developed automated systems for configuration management, but most of these systems are considered proprietary. Consequently, there is no published literature available about them.

Bersoff, E. H., "Elements of Software Configuration Management", IEEE Trans. Software Eng., V. SE-10, No.1, Jan. 1984, pp. 79-87.

Bersoff, Edward H., Henderson, Vilas D., and Siegel, Stanley G., Software Configuration Management, an Investment in Product Integrity, Prentice-Hall, 1980.

Bond, M.; Bott, M. F.; Tedd, M. D., "Saviour-A Tool for Software Configuration Management", IUCC Bull., Spring, 1983, pp. 27-29.

Clemons, E. K.; Scallan, P. G.; Sibley, E. H., "The Software Configuration Management Database", AFIPS Conference Proceedings, V. 50, 1981 National Computer Conference, 4-7 May 1981.

Diakite, L., "The ISEF Software Configuration Management System", Software Engineering, Proceedings of ESA/ESTEC Seminar, Noordwijk, Netherlands, 11-14 Oct. 1983, pp. 19-23.

Foulkes, R.; Mills, M. P., "Software Configuration Management and Its Contribution to Reliability Program Management", IEEE Trans. Reliab., V. R-32, No. 3, Aug. 1983, pp. 289-292.

Hawley, P. J., "DACOM: A Design and Configuration Management System", IEEE Proceedings COMPSAC 83: The IEEE Computer Society's Seventh International Computer Software and

Applications Conference, Chicago, IL, 7-11 Nov. 1983, pp. 580-587.

Johnson, D.; Kolberg, C.; Sinnamon, J., "A Programmable System for Software Configuration Management", Proceedings of COMPSAC 78 Computer Software and Applications Conference, Chicago, IL, 13-16 Nov. 1978, pp. 402-407.

Rasmussen, N. L., "Software Configuration Management Using Operation System Primitives of the National Software Works", 15th IEEE Computer Society International Conference, Washington, DC, 6-9 Sept. 1977.

Rochkind, Marc J., "Source Code Control System", IEEE Trans. on Software Engineering, V. SE-1, No. 4, December 1975, pp. 364-370.

Tichy, W. F., "Software Development Control Based on Module Interconnection", Proceedings of the 4th International Conference on Software Engineering, Munich, Germany, 17-19 Sept. 1979.

Wade, G. O., "AFLC Software Configuration Management", Proceedings of the IEEE 1983 National Aerospace and Electronics Conference, Dayton, OH, 17-19 May 1983.

Young, B. R., "Software Configuration Management", Software Engineering. Proceedings of ESA/ESTEC Seminar, Noordwijk, Netherlands, 11-14 Oct. 1983.

Zucker, Sandra, "Automating the Configuration Management Process", Proc. A Conference on Software Development Tools, Techniques, and Alternatives, July 25-28, 1983, Arlington, Va., IEEE Computer Society Press.

9.0 Verification and Validation

Verification and validation relate to assuring that a product conforms to requirements and user needs. When the product is a textual document, this assurance is usually in the form of a review by people who are in a position to judge the conformance of the product. The root information fragment resulting from such a review is typically a review report. The non-root fragments related to reviews are review issues (problems) and associated attributes, along with other information about the review process. When the product being verified or validated is executable, the information interfaces involved are those of testing, described above. See also quality assurance.

Campanizzi, J. A., "Structured Software Testing", Qual. Proj., V. 17, No. 5, 14-15 May 1984.

Casey, D.; Erickson, R. W., "Practical Tools for Software

Test Certification", Digest of Papers COMPCON Spring '84. Twenty-eighth IEEE Computer Society International Conference San Francisco, CA, 27 Feb - 1 March 1984, pp. 87-90.

Conway, M.; Molari, R., Guidelines for Testing and Release Procedures, Informatics General Corp., Palo Alto, CA, 15 Jan. 1984.

Dairymple, James Capt.; Forest, Steve 1LT; Forshee, Michael Capt.; Fox-Daeke, Teresa Capt.; Ingram, Guy 2LT; Papa, Dan Capt, Handbook for Evaluation and Life Cycle Planning for Software, V.4 - Test and Independent Verification and Validation, Electronic Systems Division, Air Force Systems Command, Hanscom AFB, MA, Feb. 1983.

Deutsch, M. S., Software Verification and Validation. Realistic Project Approaches, Prentice-Hall, NJ, 1982.

Evans, Michael W., Productive Software Test Management, Wiley-Interscience, 1984.

Generic Independent Verification and Validation (IV & V) Capability for OC-ALC/MME, Pram Program Office, ASD/RA, Wright-Patterson AFB, OH, Mar. 1984.

Koch, H. S.; Kubat, P., "Managing Test-Procedures to Achieve Reliable Software", IEEE Trans. Reliab., V. R-32, No. 3, Aug. 1983, pp. 299-303.

Kosovac, S. M.; Shortle, G. E. Jr., "Ballistic Missile Defense Simulation Validation", Proceedings of the 1976 Summer Computer Simulation Conference, Washington, DC, 12-14 July 1976, pp. 693-698.

Lehman, M.M., "The Environment of Program Development and Maintenance Programs, Programming and Programming Support", Proc., 1981 International Computing Symposium, IPC Business Press, Ltd., pp. 1-12.

Longoni, F.; Redaelli, R, "On-Board Software Testing and Qualification", Software Engineering. Proceedings of ESA/ESTEC Seminar, Noordwijk, Netherlands, 11-14 Oct 1983, pp. 201-206.

Pelissero, R., "Tornado Flight Control Software Validation: Methodology and Tools", Agard Conference Proceedings, NO. 330, Software for Avionics, The Hague-Kijkduin, Netherlands, 6-10 Sept. 1982, pp. 1-13.

Planning for Software Validation, Verification and Testing, National Bureau of Standards, Nov. 1982.

Reifer, D. J., Software Verification and Validation, AGARD, Rept. No. AGARD-AG-258, May 1980.

Taylor, R. N., "An Integrated Verification and Testing Environment", Software-Pract. and Exper., V. 13, No. 8, August 1983, pp. 697-713.

Wilson, P. B., "Building Quality into Software with More Effective Testing", Small Syst. World, V. 11, No. 8, Aug. 1983, pp. 42-44.

Young, N. J. B., "Automating the Testing of Software (Aerospace)", Agard Conference Proceedings No. 343. Advanced Concepts for Avionics/Weapon System Design, Development and Integration, 18-22 April 1983, pp. 1-13.

10.0 Quality Assurance

The term 'quality assurance' is used to mean different things by different people. To some, quality assurance is synonymous with validation and verification. To others, it encompasses both validation and verification and configuration management. To others, quality assurance is part of configuration management. To others, quality assurance is the process of assuring that all activities are carried out according to prescribed standards.

The information interfaces of quality assurance are, therefore, the interfaces of configuration management, validation and verification (which, in turn, are the information interfaces of reviewing and testing), and process measurement and evaluation. The reader should see all of these areas. The documents listed here are those that view the topic from the notion that quality assurance is a superset of configuration management and validation and verification.

Bergmann, S.; Gagne, P.; Paige, M., "Software QA: An Integrated Approach", Kirk, D. E. (Editor), IEEE Sixteenth ASILOMAR Conference on Circuits, Systems and Computers, Pacific Grove, CA, 8-10 Nov. 1982.

Buckley, F. J.; Poston, R, "Software Quality Assurance", IEEE Trans. Software Eng., V. SE-10, No. 1, Jan. 1984, pp. 36-41.

Butler, L. P., "Software Quality Assurance Cyclomatic Complexity of a Computer Program", IEEE Proceedings of the IEEE 1983 National Aerospace and Electronics Conference, NAECON 1983, V. 2, 17-19 May 1983, pp. 867-73.

Carpenter, C. L. Jr.; Murine, G. E., "Measuring Software Product Quality", Qual. Prog., V. 17, No. 5, 16-20 May 1984.

Carpenter, L.C., and Tripp, L.L., "Software Design Validation Tool," Proceedings of the 1975 International Confrence on Reliable Software, IEEE Computer Society, 1975, pp. 395-400.

Catiglione, P. V.; Thompson, W. W., "Implementation and Measurable Output of Software Quality Assurance", 1983 Proceedings Annual Reliability and Maintainability Symposium, Orlando, FL, 25-27 Jan. 1983, pp. 107-112.

Crawford, S. G.; Hiering, V. S., "Software Quality Control and Assurance", IEEE International Conference on Communications, 1983, pp. 713-717.

Greenberg, S. G., "A Systems Approach to Software Quality Assurance", Second Annual Phoenix Conference on Computers and Communications. 1983 Proceedings, Phoenix, AZ, 14-16 March 1983, pp. 180-184.

Kitchenham, B. A., "Program History Records: A System of Software Data Collection and Analysis", ICL Tech J., vol. 4, no. 1, 1 May 1984, pp. 103-114.

McCall, Jim A.; Richards, Paul K.; Walters, Gene F., Factors in Software Quality, Volume I. Concepts and Definitions of Software Quality, General Electric Co., Sunnyvale, CA, Report No. RADC-TR-77-369-VOL-1, Nov. 1977.

McCall, Jim A.; Richards, Paul K.; Walters, Gene F, Factors in Software Quality. Volume III. Metric Data Collection and Validation, General Electric Co., Sunnyvale, CA, Report No. RADC-TR-77-369-VOL-2, Nov. 1977.

Murine, G. E., "The Application of Software Quality Metrics", IEEE Second Annual Phoenix Conference on Computers and Communications, 1983 Conference Proceedings, Phoenix, AZ, 14-16 March 1983, pp. 185-188.

O'Connell, G. S., "Software Quality Assurance, A General Approach", Second Software Engineering Standards Application Workshop, San Francisco, CA, 17-19 May 1983, pp. 96-102.

Perry, William E., Effective Methods of EDP Quality Assurance, Prentice-Hall, 1983.

Pingel, T. C., "Assuring Quality in a Telecommunications Software Development Project", IEEE International Conference on Communications: Integrating Communication for World Progress (ICC '83, V. 2, Boston, MA, 19-22 June 1983, pp. 718-721.

Proceedings of the Workshop on Product Assurance Techniques for Embedded Computer Systems Held at White Oak, Silver Spring, MD, Naval Surface Weapons Center White Oak, Silver Spring, MD, 11-12 Jan. 1984.

Pyper, W. R., "Historical Files for Software Quality Assurance", NBS Proceedings of the Computer Performance Evaluation Users Group (CPEUG) 17th Annual Meeting

(NBS-SP-500-83) San Antonio, TX, 16-19 Nov. 1981. pp. 101-106.

Thomas, E. F., "Pitfalls of Software Quality Assurance Management", 1983 Proceedings Annual Reliability and Maintainability Symposium, Orlando, FL, 25-27 Jan. 1983, pp. 101-106.

Tice, G. D. Jr., "SQA Contributions to a Quality Software Product, 1984 Proceedings of the Annual Reliability and Maintainability Symposium, San Francisco, CA, 24-26 Jan. 1984, pp. 537-539.

11.0 Systems (Environment) Management

Our interest in systems, or operations, management is related to the operation of the JSSEE itself. In particular, with respect to this study we are interested in those information interfaces created by the need to manage the use of a JSSEE in an operational setting. Typical needs include system performance evaluation and usage accounting. The following references pertain to performance evaluation.

Ferrari, Domenico, ed., Performance of Computer Installations, Elsevier/North-Holland, 1978.

Ferrari, Domenico, Computer Systems Performance Evaluation, Prentice-Hall, 1978.

Literature sources related to other areas of operations management needs and information interfaces were not readily available during the limited time frame of this study. Consequently, this paragraph is intended to serve as a reminder that this subject needs further examination to identify what ways a SEE should provide data to support system management. The following is one possible example.

Frobose, R. H.; Wilgus, C. A., "On-Line Entry and Analysis of Computer Center Operations Logs", Aesop Operations Managers' Conference, Las Vegas, NV, 4 May 1983, Report No.: UCRL-88736; CONF-830536-1.

12.0 Training

The Operational Concept Definition for the JSSEE identifies several ways the JSSEE will support training activities, including:

* Software development and maintenance (adapting the MCCR and its documentation, building special training systems, etc.)

* Preparing instructional materials (computer-assisted and other)

* Managing the instructional process (scheduling classes, tracking performance, etc.)

* Scenario development and data reduction

For the purposes of this report, we view the software development and maintenance needs of training as extensions of the development and maintenance needs of MCCR software. The other areas of training support create special needs because they are applications which run on the JSSEE itself. The information interfaces created by CAI systems, classroom management systems and scenario and data handling systems are legitimate concerns for the JSSEE developers. We have not had sufficient time to explore these areas in much depth. The following references can serve as a start.

Crider, Janet and Wagner, Carmen, CAI Guide to Courseware Languages, Dilithium Press, 1985.

Kearsley, Greg, Computer-based Training: A Guide to Selection and Implementation, Addison-Wesley, 1983.

Schwartz, J., "Languages and Systems for Computer-Aided Instruction", Machine-Mediated Learning, Vol. 1, No. 1, 1983, pp. 5-39.

13.0 Office Automation and Word Processing

Software engineering environments are considered special purpose office automation systems, where text handling (including mixed text and graphics), documentation and office management are important concerns. Consequently, the office automation community has much to say that is of interest to the designers of the JSSEE.

Ahlsen, Matts; Bjornerstedt, Anders; Britts, Stefan; Hulten, Christen; Soderlund, Lars; "An Architecture for Object Management in OIS", ACM Trans. on Office Information Systems, V. 2, No. 3, July 1984, pp. 173-196.

Gilder, Jules H., The Integrated Software Book, Addison-Wesley, 1985.

Lynback, Peter; McLeod, Dennis; "Object Management in Distributed Information Systems", ACM Trans. on Office Information Systems, V. 2, No. 2, April 1984, pp. 96-122.

Yao, S. Bing; Henner, Alan R.; Shi, Zhougzhi; Tuo, Dawei; "FORMANAGER: An Office Forms Management System", ACM Trans. on Office Information Systems, V.2, No. 3, July 1984, pp. 235-262.

14.0 Networking and Distributed Processing

A software engineering environment may be a distributed system. In such a case, the fields of networking and distributed processing may be fruitful areas to search for information interfaces that need to be considered in the design of the JSSEE. These fields are also likely application areas for systems built using the JSSEE, and therefore, should be examined at least from that perspective. Both viewpoints are represented in this section; there is no separate description of these areas under Applications.

Berg, Helmut K.; Paulsen, William R.; Wood, William T.; Yu, Stone H., "Concurrent System Description Language", (Final Technical Rept. 11 Sept. 80-11 Sept. 81) Corporate Sciences Center, Report No. RADC-TR-82-3, February 1982.

Brinch-Hanson, P., Architecture of Concurrent Programs, Prentice-Hall, 1977.

Cotronis, J. Y.; Lauer, P. E., Shields, M. W., "Formal Behavioural Specification of Concurrent Systems without Globality Assumptions", Computing Lab. Corp., Report No. UNT/CL/TRS-162, 1981.

Martin, James, Design and Strategy for Distributed Data Processing, Prentice-Hall, 1981.

Muntz.C., Marcus, M., Sattley, K., and Shipman, B., NSW (National Software Works) Lessons Learned, RADC-TR-84-90, Massachusetts Computer Associates, Wakefield, MA., May 1984.

Tanenbaum, Andrew S., Computer Networks: Toward Distributed Processing Systems, Prentice-Hall, 1981.

Vernon, Mary Katherine, Performance-Oriented Design of Distributed Systems, UCLA Computer Science Dept., 1982.

Weber, Herbert, "The Distributed Development System - A Monolithic Software Development Environment", Software Engineering Notes, v. 9, No. 5, Oct. 1984, ACM SIGSOFT.

Wileden, Jack C., "Dream -- An Approach to Designing Large Scale, Concurrent Software Systems", Proc Annu Conf ACM, Detroit, MI, Oct. 29-31, 1979, Publ. by ACM, Baltimore, MD 1979, pp. 88-94.

15.0 Graphics

Our interest in graphics here is related to the need for the JSSEE to support graphic representations of program and data

structures.

Belady, L.A., Evangelisti, C.J., and Power, L.R., "GREENPRINT: A Graphic Representation of Structured Programs," IBM Systems Journal, Vol. 19, No. 4, 1980, pp. 542-53.

Hebalkar, P.G. and Zilles, S.N., TELL: A System for Graphically Representing Software Design, IBM Corp., Research Report RJ2351, September 1978.

Panasuk, Curtis, "Software Standards will Usher in the Age of Graphics", Electronic Design, July 12, 1984.

Wein, Marceli (ed.), "Proceedings - Graphics Interface '83, 1983", National Research Council of Canada, Ottawa, Ont. Can., Proc Graphics Interface '83, Edmonton, Alberta, Can.

16.0 Future Paradigms

The list of subjects that can be thought of as future paradigms that might become a part of the JSSEE someday is open ended. In fact, the JSSEE is defined as a mid-range system not necessarily encompassing any of the topics covered in the references below, but needing to accommodate the evolution into whatever directions turn out to be particularly useful at a later date. As such, the JSSEE does not have to accommodate any information interfaces related to these future paradigms now. It just has to be able to expand later. The references listed here can provide insight into some of the directions this might take.

16.1 Artificial Intelligence and Logic Programming

Artemsevh, I.L.; Gorbachev, S.B.; Kleshchev, A.S.; Lifshits, A. Ya; Orluv, S.I.; Orluva, L.D.; Ovhrovh, T.G.; "Compiler Generator for Knowledge Representation Languages", Programmirovanie (USSR), v. 9, no. 4, July-Aug. 1983, pp. 78-89.

Ballard, B.W.; Lusth, J.C.; Tinkham, N.L., "LDC-1: A Transportable, Knowledge-Based Natural Language Processor for Office Environments", ACM Trans. Off. Inf. Syst., v. 2, No. 1, Jan. 1984, pp. 1-25.

Balzer, Robert; Cheatham, Thomas E. Jr; Green, Cordell; "Software Technology in the 1990's: Using a New Paradigm", Computer, v. 16, No. 11, Nov. 1983, IEEE Computer Society.

Barr, Avron, and Feigenbaum, Edward A., The Handbook of Artificial Intelligence, William Kaufmann Inc., 1981.

Billion, J.P., Comparison of Languages for Artificial Intelligence, Productivity and Data Profession: Two

Essentials for a Dynamic Company. Proceedings of the Spring Convention, v.1, 30 May - 3 June 1983, Paris, France, pp. 250-254.

Clocksin, W.F., Mellish, C.S., Programming in PROLOG, Springer-Verlag, 1981.

Dean, J.S.; McCone B.P., "Trends for Advanced Software Tools", EASCON '83: 16th Annual IEEE Electronics and Aerospace Systems Conference and Exposition, Washington, DC, 19-21 Sept. 1983, pp. 291-298.

Dean, J.S.; McCone, B.P.; Shapiro, D.G., "A Knowledge Base for Supporting An Intelligent Program Editor", Proceedings of the 7th International Conference on Software Engineering, Orlando, FL, 26-29 March 1984.

Fidge, C.J.; Cain, G.J; Jackson, L.N.; Fascoe, R.S.V., "Overview of the Melba Automatic Code Generation Project", ATR Aust Telecommun. Res., Vol. 18, No. 1, 1984, pp. 3-12.

Greenspan, Sol J., "Requirements Modeling: A Knowledge Representation Approach to Software Definition", CSRG Technical Report No. 155, Univ. of Toronto, Computer Systems Research Group, n.d.

Kitagawa, I.; Mizoguchi, F., "A Software Environment for Developing Knowledge Base Systems", Computer Science and Technologies 1982, 1982, pp. 334-339.

Merry, M., "Apex 3: An Expert System Shell for Fault Diagnosis", GEC J. Res. Incorp. Rootni Rev. (GB), v. 1, No. 1, 1983, pp. 39-47.

Rawlings, T.L., "A Discussion of Knowledge Representation Within the Darts Technology", IEEE Conference Record of the 17th ASILOMAR Conference on Circuits, Systems and Computers, Pacific Grove, CA, 31 Oct. - 2 Nov. 1983. pp. 1-3.

Rich, C.; Waters, R., "Computer Aided Evolutionary Design for Software Engineering", Sigart Newsl., No. 76, April 1981, pp. 14-15.

Rich, Elaine, "Recent Applications of Artificial Intelligence in Programming and Problem Solving", Computer Magazine, v. 17, No. 5, May 1984, pp. 4-12.

Steele, Guy L., COMMON LISP: The Language, Digital Press, 1984.

Waters, Richard C., "The Programmers Apprentice: Knowledge Based Program Editing", IEEE Trans. on Software Engineering, V. SE-8,No. 1, January 1982, pp. 1-12.

16.2 Application Generators

Horowitz, Ellis; Kemper, Alfous; Narasimhan, Balagi; "A Survey of Application Generators", IEEE Software, v. 2, No. 1, January 1985, pp. 40-54.

Martin, James, Application Development Without Programmers, Prentice-Hall, 1982.

Waldrop, J.H., Application Generators: A Case Study, AFIPS Press.

16.3 Functional Programming

Backus, John, "Can Programming be liberated from the von Newmann Style? A Functional Style and Its Algebra of Programs", Comm. ACM, v. 21, No. 8, Aug. 1978.

Zave, Pamela, "An Operational Approach to Requirements Specification for Embedded Systems", IEEE Trans. on Software Engineering, Vol. SE-8, No. 3, May 1982, pp. 250-269.

16.4 Relational Programming

Best, E., Relational Semantics of Concurrent Programs (with Some Applications), (Technical Rept. Series), Newcastle Upon Tyne Univ. (England) Computing Lab., Report No. NTU/CL/TRS-180, 1982.

MacLennan, Bruce J., A Relational Program for a Syntax Directed Editor (Technical Report), Naval Postgraduate School, Monterey, CA, Report No. NPS52-82-006, Apr. 1982.

MacLennan, Bruce J., Introduction to Relational Programming, (Technical Report), Naval Postgraduate School, Monterey, CA, Report No. NPS52-81-008, June 1981.

16.5 Rapid Prototyping

Rapid prototyping literature can also be found under the headings related to the specific approaches used to build prototypes. Prototyping through reuse of existing code is covered in Section 3.2 above related to design. Rapid prototyping through the use of application generators is covered in Section 16.3 above. The "Special Issue on Rapid Prototyping", below, contains 30 working papers on the subject from a variety of approaches.

The Development of a Programming Support System for Rapid Prototyping, Software Options, Inc., Report No. SO-01-83, 1983

Konchan,Thomas, and Klausner, Aviel, Rapid Prototyping and Requirements Specification Using PDS, TR-02-83, Harvard University, Center for Research in Computing Technology, n.d.

"Special Issue on Rapid Prototyping", Software Engineering Notes, Vol. 7, No. 5, December 1982.

Zelkowitz, Marvin V., "A Case Study in Rapid Prototyping", Software Practice and Experience, Vol 10, 1980, pp. 1037-1042.

16.6 Data Flow Computing

Ackerman, W.B., and Dennis, J.B., "VAL - A Value Oriented Algorithmic Language", Laboratory for Computer Science, MIT, Preliminary Reference Manual, June 1979.

Arvind, Gostelow, K.P., and Plouffe, W.E., "An Asychronous Programming Language and Computing Machine", Department of Information and Computer Science, U. of California Irvine, TR 114A, September 1978.

McGraw, J., "Data Flow Computing: Software Development", IEEE Trans. on Computing, Vol. 29, No. 12, December 1980, pp. 1095-1103.

17.0 Languages and Syntax-Directed Processing

The subject of languages and language processing suggest a number of information interfaces that are relevant to the JSSEE. Most of these information interfaces are internal, not usually seen directly by the environment user, although in syntax-directed editors they are frequently represented by the entities that the user manipulates.

The following document discusses the impact of language technology on software engineering.

Davies, A.C., Programming Language Features Which Support Software Engineering Design Methods, Centre of Information Engng., City Univ. of London, London, England, DePledge, P. (Editor), Software Engineering for Non-rootprocessor Systems, Publ. Peter Peregrinus, London, England 1984.

The following documents discuss information interfaces related to syntax-directed editing.

Barrow, A; Grace,E., "Slash Software Development - A Case for Incremental Design", Proceedings of the IEEE 1983 National Aerospace and Electronics Conference, NAECON, 17-19 May 1983, pp. 973-978.

Barstow, David R.; Shrobe, Howard E.; Sandewall, Erik, Interactive Programming Environments, McGraw-Hill, 1984.

Medina-Mora, Raul, Syntax-Directed Editing: Towards

Integrated Programming Environments, Ph.D. Dissertation, Carnegie-Mellon University, March 1982.

Reps, Thomas W., Generating Language-Based Environments, MIT Press, 1984.

Hardware description languages may be included in the JSSEE as part of the need to support the systems engineering activity. In any case, they represent a class of languages that have some interesting capabilities, particularly in the area of graphics, that may be of interest to the JSSEE. The following references reflect some recent developments in this field.

Computer, Vol. 18, No. 2, February 1985. Special issue on hardware description languages; contains eight articles on the subject.

Lieberherr, Karl J., "Toward a Standard Hardware Description Language", IEEE Design and Test of Computers, Vol. 2, No. 1, February 1985.

18.0 Applications

It seems that there has been at least one methodology or language developed for just about every class of applications for which computers are used. Each of these presents a specific list of information interfaces appropriate to an environment being used for all applications viewed as mission critical, which encompasses a very broad range of applications. It is appropriate, therefore, to consider the implications of these applications in discussing the information interface requirements of JSSEE. In this section we provide a small sampling of literature which is of an application specific nature, more as a prod to the imagination than as a thorough covering of the subject.

The application specific area consists of topics with computer science names, such as distributed database management systems, and end user applications, such as avionics.

18.1 Database Management Systems

This subject is being covered in greater detail in a separate study.

Batory, D.S.; Model of Transaction on Physical Databases, Report No.: DOE/ER/10977-T1, Dept. of Energy, 1982.

Bachman, C.W., "Data Structure Diagrams", Data Base, The Quarterly Newsletter of the Special Interest Group on Business Data Processing of the ACM, Vol 1, No. 2, pp. 4-10.

Chen, P., The Entity-Relationship to Logical Data Base Design, The Q.E.D. Monograph Series on Data Base Management, No. 6 (Wellesley, Mass.: Q.E.D. Information Sciences, Inc.), 1977.

Haseman, W.D.; Whinston, A.B., "Automatic Application Program Interface to a Data Base", Comput J., Vol. 20, No. 3, August 1977, pp. 222-226.

Jardine, D.A.; Davis, B.J., "A Data-Base Application Design Language", Inf. and Manage., Vol. 4, No. 2, May 1981, pp. 81-93.

Knoll, Matthew; Hargrave, W. Terry; Salazar, Sandra, "Data Model Processing", Proceedings of the National Computer Conference, Houston, TX, June 7-10, 1982, pp. 571-578.

Lamersdorf, Winfried, Specification and Interpretation of Data Model Semantics: An Integration of Two Approaches, Report No.: NBSIR-83-2740, National Bureau of Standards, July 1983.

Martin, J., Principles of Data-Base Management, 2nd Ed., Prentice-Hall, 1977.

Martin, J., Strategic Data Planning Methodologies, Prentice-Hall, 1982.

Nakamura, A., "Three-Valued Logic and Its Application to the Query Language of Incomplete Information", Proceedings of the International Symposium on Multiple-Valued Logic (13th), held at Kyoto, Japan on May 23-25, 1983, pp. 214-218.

Niemi, T.; Jarvelin, K., "A Straightforward Formalization of the Relational Model", Sigmod Rec., Vol 14, No. 1, March 1984, pp. 15-38.

Olle, T. William, "The Current Programming Language Standards Scene IX: Data Base Management Systems", Computer & Standards, V. 2, No. 2-3, 1983, North-Holland, pp. 119-126.

Parsons, R.G.; Dale, A.G.; Yurkanan, C.V., "Data Manipulation Language Requirements for Database Management Systems", Comput J., Vol. 17, No. 2, May 2, 1974, pp. 99-103.

Sneeringer, Cheryl, "Lolipop: A Data Manipulation Language for Data-Independence", Proc. of the Tex. Conf. on Comput. Syst., 5th, Univ. of Tex, Austin, Oct 18-19, 1976, pp. 140-146.

Ulfsby, Stig; Meen, Steinar; Oian, Jorn, "Tornado: A Data-Base Management System for Graphics Applications", IEEE Comput Graphics Appl, Vol. 2, No. 3, May 1982, pp. 71-76, 78-79.

Wah, Benjamin W., "International Conference on Data Engineering, 1984", IEEE Service Center (Cat. No. 84CH2031-3).

18.2 Avionics

Carrier, L. M.; Kasai, G. H., "Avionics Software Management and Control", Proceedings of the IEEE/AIAA 5th Digital Avionics Systems Conference, Seattle, WA, 31 Oct. - 3 Nov. 1983.

Weiss, David, NATO/AGARD Symposium on Software for Avionics, Naval Research Lab., Washington, DC 20390, Agency Report No.: C-7-83, April 1983.

18.3 Decision Support Systems

Jones, P., "Reveal-Addressing the Technologies of DSS and Expert Systems", IEEE Colloquium on Decision Support Aspects of Expert Systems (Digest No. 67), 1984.

Orman, L., "An Array Theoretic Specification Environment for the Design of Desicion Support Systems", Int. J. Policy Anal. and Inf. Syst., v. 6, No. 4, Dec. 1982.

Wang, M.S.-Y; Yu, K.-C., "A Hierarchical Skeleton of Knowledge-Based Decision Support System Software", Proceedings COMPCON 83 Fall: Delivering Computer Power to End Users, Twenty-Seventh IEEE Computer Society International Conference, 25029 Sept. 1983, Arlington, VA, October 1983, pp. 86-92.

18.4 Real-Time Systems

Biewald, J.; Joho, E.; Jovalakic, S.; Shelling, H., "Application of the Specification and Design Technique EPOS to a Process Control Problem", Proc of the IFAC/IFIP Conf. 6th, Duesseldorf, Germany, Oct. 14-17, 1980. Publ forIFAC by Pergamon Press, Oxford, England and New York, NY 1980, pp. 517-522.

Glass, Robert, Real-Time Software, Prentice-Hall, 1984.

Gomaa, H., "A Software Design Method for Real-Time Systems", Comm. ACM, V.27, No. 9, September 1984, pp. 938-949.

Mellichamp, Duncan A., Real Time Computing: With Applications to Data Acquisition and Control, Van Nostrand Reinhold, 1983.

Weide, Bruce W., et al., "Integration and Design Methodology Support for Process Control", Computer Magazine, v. 17, No. 2, February 1984, pp. 27-32.

18.5 Human Engineering

Not strictly an application, human engineering is an aspect of all applications. This topic is being discussed in greater depth in a seperate study. Of the following references, the first three provide good overviews of the area; the last three concern tools and languages.

Curtis, B., Tutorial: Human Factors in Software Development, IEEE Computer Society Press, Los Alamitos, Calif., 1981.

Shneiderman, B., Software Psychology: Human Factors in Computer and Information Systems, Little Brown and Co., Boston, Mass., 1980.

Weinberg, G.M., The Psychology of Computer Programming, Van Nostrand Reinhold, New York, 1971.

Hartson, H.R., and Johnson, D.H., "Dialogue Management: New Concepts in Human-Computer Interface Development". Manuscript submitted to ACM Computing Surveys (revised version), March 1984.

Moran, T.P., "The Command Language Grammar: A Representation of the User Interface of Interactive Computer Systems". International Journal of Man-Machine Studies, 1981, Vol. 15, pp. 3-50.

Reisner, P., "Formal Grammar and Human Factors Design of an Interactive Graphics System". IEEE Transactions on Software Engineering, Vol. SE-7, 1981, pp. 229-240.

The following reference discusses the human interface in a number of recent microcomputer products.

Gilder, Jules H., The Integrated Software Book, Addison-Wesley, 1985.

18.6 Security

This subject is being covered in greater depth in a separate study. We list it here as a reminder that security is a concern of most mission-critical software. The following references may serve as a starting point for further study.

Cheheyl, M.H., et al., "Verifying Security", ACM Computing Surveys, Vol. 13, No. 3, September 1981.

Gerhart, S.L., ed., "AFFIRM User's Guide", USC Information Sciences Institute, April 1980.

Hartman, Bret A., et al., "Formal Verification in the Trusted System Evlauation Process", Conference Record of EASCON 82, Washington, DC, September 1982.

Landwehr, C.E., "Formal Models for Computer Security", ACM Computing Surveys, Vol. 13, No. 3, September 1981.

Levitt, K.N., et al., "The HDM Handbook, Vol. I-III", Computer Science Laboratory, SRI International, Menlo Park, CA, June 1979.

Locasso, R., et al., "The Ina Jo Specification Language Reference Manual", SDC Document TM-(1)-6021/001/00, System Development Corporation, Santa Monica, CA, June 1980.

Millen, J.K., "Security Kernal Validation in Practice", Communications of the ACM, Vol. 9, No. 5, May 1976.

19.0 Relationships

The following report is the only one we found that explicitly discusses the relationships between information interfaces. In particular, it contains a discussion of producer-consumer relationships. Some relationships are identified in almost every reference in this report. Typically these are structural relationships (such as component-subcomponent) or temporal (producer-consumer). Most methodologies, of course, describe both time sequences and structures of work products, but they usually just present these relationships in a prescriptive way, without analyzing them. Nevertheless, for the purposes of this study, relationships among information fragments can be found almost everywhere that the information interfaces, themselves, can be found.

Santoni, Patricia A., The Project Development Data Base: The Core of an Automated Software Engineering Environment, Naval Ocean Systems Center, Technical Note 932, 24 October 1980.

Environment builders are becoming increasingly concerned with

the definition of a project database as one of the primary focuses of the environment. Appendix 1 is taken from such an effort.

There are innumerable relationships specific to individual tools and information fragments. We can suggest the following to provide a flavor of what a researcher should be looking for:

* The "with" and "uses" relationships in Ada
* Data flow and scope relationships related to programs
* Symbol cross references, including document indexes
* Actual-expected relationship in test results

20.0 Conclusions

The main purpose of this report is to survey existing information interfaces and information interface technology, providing references to literature on these topics and offering representative examples of both information interfaces and information interface technology. In this section we summarize that survey by listing what we have described in the report. The following lists assemble the information interfaces and the interface technology that we have described in the tools review sections of this report.

Information Interfaces:

- RSL, the DCDS (SREM) Requirements Statement Language [section 2.1.1]
- SREM's element-attribute-relation structure [section 2.1.1]
- DCDS tool-interpretation tables [section 2.1.1]
- Arcturus interactive Ada program design language (PDL) [section 3.1.1]
- Arcturus interactive Ada [section 4.1.1]
- Arcturus internal representation [section 4.1.1]
- compiled version of an (interactive or otherwise) Ada program [section 4.1.1]
- Arcturus performance measurement output [section 4.1.1]
- Toolpack program units and program unit groups (PUGs) [section 4.1.2]
- token streams, parse trees, symbol tables and object modules [section 4.1.2]

- Toolpack test data collections (TDCs) and error reports [section 4.1.2]

- annotated flow graphs and versions of programs instrumented to support debugging [section 4.1.2]

- Toolpack options packets and procedures [section 4.1.2]

- Diana attributed trees [section 4.1.4]

- EDL debugging viewpoint definitions [section 4.1.5]

- CTS problem report and tracking form [section 6.1.1]

- statistical summary of CTS activities [section 6.1.1]

- EPEST cost estimation charts, weekly group progress reports, milestone charts, and activity schedules [section 7.1.1]

- SMSS access control records [section 8.1.1]

- SMSS transactions log [section 8.1.1]

- SMSS source code modification records (deltas) [section 8.1.1]

- listings, error reports and load modules [section 8.1.1]

Information Interface Technology:

- Odin [section 4.1.3]

- Interface Definition Language [section 4.1.4]

- PIC internal representation definition tools [section 3.1.2]

- Standard language definition methods (e.g., BNF) [found in various tool specifications, e.g., Arcturus PDL definition]

- Data dictionaries

The main conclusion to be drawn from the preceding lists is that there are a vast array of information interfaces that must be considered in designing a software development environment and relatively little information interface technology to help in the task of designing, creating and controlling those information interfaces.

Another conclusion that falls out from the enormous breadth of the subject is that there is still a lot of work remaining (as

described in Section 21 below). In order to actually develop information interface specifications, some number of existing information interfaces should be disected in detail.

Given the limited time available, this compendium is necessarily incomplete. Nevertheless, as the lists above suggest, it has succeeded in graphically demonstrating the scope of the information interfaces issue relative to the creation of a JSSEE. It has also indicated the paucity of information interface technology available for coping with this issue. Although vastly more citations could have been accumulated, we believe that those contained in this compendium are reasonably representative of the topic area. Moreover, we feel that this compendium should provide an excellent starting point for any subsequent efforts addressing the information interfaces issue with respect to JSSEE.

21.0 Recommendations

It should be clear that an effort of this type could go on almost indefinitely. While we do not recommend that, we do believe that some additional work is needed. Specifically, a closer look at each of the areas should be made in an effort to distinguish between the important (for the JSSEE) and the less important references, and to develop a list of the basic concepts of each area. The objective here would be to further bound the problems that will need to be examined to select specific information interfaces for the JSSEE in a later task. We are not ready to do the selecting yet, but we are ready to start narrowing the field of potential interfaces that the JSSEE should support.

Several other steps need to be completed before a list of information interfaces can be prescribed for the JSSEE:

- All relevant standards need to be detailed.

- A framework for organizing information interfaces must be developed and a more formal taxonomy of information interface related concepts needs to be defined.

- The basic notion that a single environment can support multiple methods and approaches to development in a consistent way needs to be established.

- The initial list of methods, languages, tools and notations that the JSSEE will support should be defined.

- A greater analysis of the Services-related environments (AIE, ALS, ALS/N, FASP and DCDS), as well as the most common commercial environments in use by DoD contractors, should be conducted with an eye toward identifying those information interfaces that will be necessary to include in the JSSEE in order to support a smooth transition from Service-specific and organization-specific environments to a Joint Services

environment.

Finally, a technology should be adopted for specifying information interfaces, e.g. a formal specification language. A decision needs to be made as to whether a standard for an information interface specification language should be adopted. It may be appropriate to develop or select tools to aid in developing, adopting, disseminating and enforcing information interface specifications.

Current JSSEE plans call for most of these steps to be taken. The effort leading to this report has reemphasized the large breadth of information interface concerns and potential contents. Substantial work remains before the actual detailed effort to specify interfaces can begin, and the specification effort will clearly be very substantial.

In fact, one final recommendation is for the planning for the JSSEE to include establishing a mechanism for the adoption of standard information interfaces after initial development of the JSSEE. Since an information interface is a shared boundary, the need for an interface specification does not really exist until the entities which form the boundary are to be specified. Naturally, many such boundaries can be anticipated; this is the basis for having a task for defining information interfaces. However, it is obvious that the complete specification of information interfaces will not be achieved in one step. The recommendations above call for the development of the technology for defining new information interfaces. This recommendation calls for the planning of the process.

APPENDIX 1

The TRW Software Productivity System

Ann B. Marmor-Squires

The TRW Software Productivity System (SPS) (Boehm, 1982) (Boehm, 1984) is a software support environment based upon the UNIX(*) operating system, a variety of TRW software tools and computer support equipment, and a wideband local network. It is part of a major TRW internally-funded effort begun in 1981 to improve productivity of software developers in the 1980s. Two of the primary components of the software development environment are relevant to information interfaces: the software tool set and the master project database. In addition, an internal research and development project is underway at TRW to define, design and develop an integrated project master database for SPS.

The goal of the software tool set is to support the entire software development lifecycle and to be well-engineered as well as portable across various hardware environments. The tool set is to provide convenient access to the master project database which ideally would contain all information relevant to project activities, including budget, personnel, schedules, management data, software requirements, design, test procedures and code. The three categories of software supported by SPS are (1) general utilities, (2) office automation tools (e.g., word processing, forms management, electronic mail, calendar management tools) and (3) software development tools (e.g., automated unit development folder, requirements traceability tool, Caine-Farber-Gordon PDL, Ada PDL, FORTRAN 77 analyzer).

In the current SPS master project database, a multi-database support structure was chosen consisting of:

1. A hierarchial file system for the software information fragments (plans, specifications, standrds, code, data, manuals, etc.) -- provided by the UNIX file system.

2. An update-tracking system for representing the successive updates of each information fragment -- provided by the UNIX Source Code Control System (SCCS).

3. A relational DBMS for representing the relations between information fragments -- currently provided by the INGRES relational DBMS.

* UNIX is a trademark of Bell Laboratories.

An advantage of this approach is that each information fragment is stored only once and updated in only one place. The relational DBMS handles the fact that some information fragments are part of several larger ones. For example, the design of a module can simultaneously be part of the system design specification as well as part of a software engineer's Unit Development Folder.

A high-level view of the current master project database structure is shown in Figure 1. On the left are the work products that are generated by the project such as specifications, code, manuals, reports and other documentation. On the right are the various classes of resources required to develop the work products such as labor dollars, capital, personnel and computer resources. In the center are the various plans linking the resources to the creation of the work products. The upper part of the figure shows the various attributes of the entities, such as architectural relationships or traceability relationships.

Although the details of this master project database structure are still in various stages of definition, a working prototype of the structure has been developed. Some portions have been worked out in sufficient detail to support the development of key tools (e.g., the Requirements Traceability Tool and the Unit Development Folder). Other portions, such as the entities, attributes and relationships of a hardware-software architecture are still under research and development.

The current master project database organization has provided a stable workable base for configuration management. SCCS controls all the baselined work products (such as source code, manual pages, user's manuals) through an Electronic Maintenance Folder (EMF). All software produced is stored in a single UNIX directory. Each EMF directory is subdivided into separate subdirectories for source code, documentation, manual pages, test information, requirements and design. Control procedures supported by SCCS allow developers and managers access to and update of a document without affecting the official baselined copy. There is also the ability to recover earlier versions of a controlled document.

The definition, design and development of an integrated project master database for SPS is currently underway as an internal R&D effort. In 1984, the project identified the components of a project master database, their characteristics and the relationships among those components. The components and relationships are described in the Project Database Model (Penedo, 1984). An important design principle has been the use of a unified conceptual model for the user's interaction with the system as an object-oriented approach. Specifically, an entity-relationship type model is used to represent the data. (This is currently considered the best available but may not be sufficient -- an important issue for further research.) The

Project Database Model consists of 30 objects, approximately 220 attributes and approximately 110 relationships.

Objects are the components of the model which characterize types of data relevant to a project. The objects are: accountable task, change item, consumable purchase, contract, data component, dictionary, document, equipment purchase, external component, hardware architecture, hardware component, hardware component description, interface, milestone, operational scenario, person, problem report, product, product description, requirement, resource, risk, simulation, software component, software executable task, software purchase, test case, test procedure, tool and Work Breakdown Structure element.

Attributes characterize the objects. Each object has a set of attributes associated with it and each instance of an object will have different values for those attributes. Relationships represent relations between objects showing their interconnectivity. An example of the attributes for the object accountable task is in Figure 2. The relationships associated with the object are shown in Figure 3.

In 1985, this R&D project is synthesizing and analyzing the methods and procedures associated with the database components as well as prototyping a subset of the project master database. It is exploring several of the issues relevant to information interface technology such as: (1) appropriate data model suitable for representing the database components, relationships, mechanisms and views; (2) need for mechanisms for interfacing with external databases; (3) need for database extensibility; (4) need for efficient mechanisms for collection and retention of historical data.

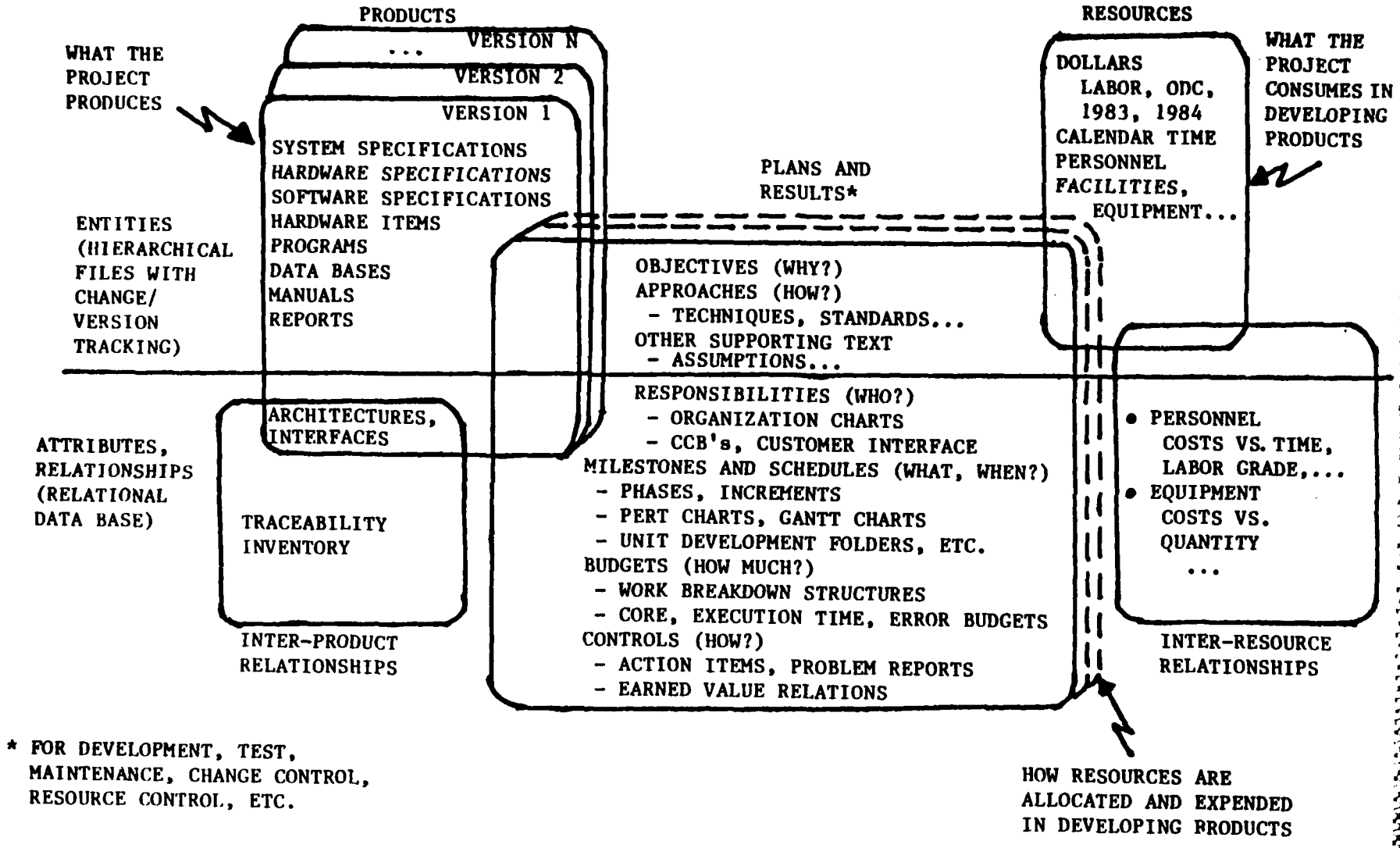
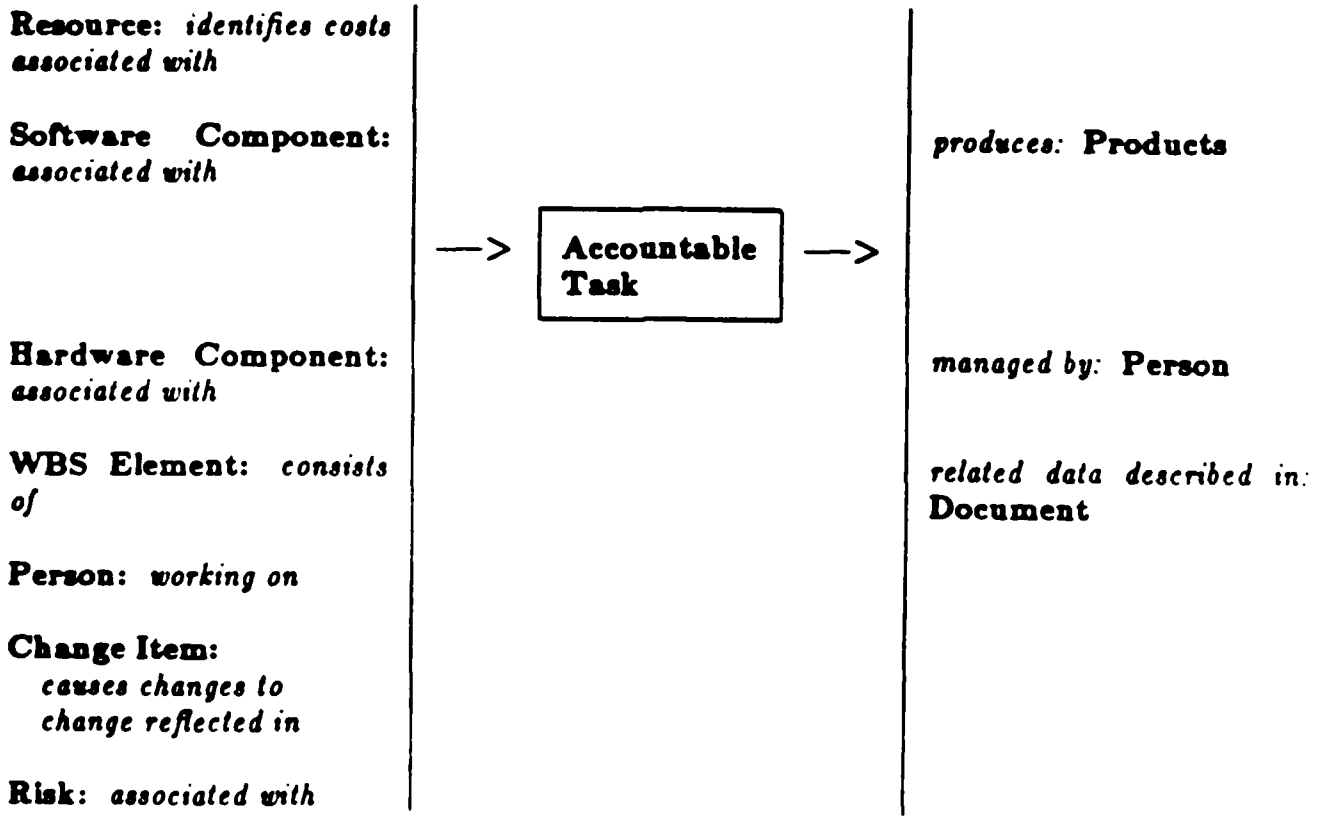


Figure 1. High-Level View of Master Project Database

Figure 2 - Accountable Task Attributes	
Name	**Description**
name	the title/name of the accountable task.
number	the job number which is used for cost purposes.
description	any textual data describing the accountable task.
period of performance	first month/year and total number of months for which this Accountable Task will be performed.
cost	estimated and actual cost per month for period of performance
Basis of Estimate	description of the basis of cost estimation.
designation	type or designation to which the job/task applies, i.e., recurring/non-recurring, etc.
type	flag to indicate whether a task/job is measurable or a level of effort.
number of people	estimated and actual number of people per labor category per burden pool per month.
internal organization	name of internal organization, if responsibility for the task lies with an internal organization, not a person.

Figure 3 - Object *Accountable Task* and its Relationships



REFERENCES

Boehm, Barry W. et al, "The TRW Software Productivity System", Proceedings of the 6th International Conference on Software Engineering, Tokyo, Japan, September 1982.

Boehm, Barry W. et al, "A Software Environment for Improving Productivity", IEEE Computer, June 1984.

Penedo, Maria H. and Stuckle, E.D., Final Report on the Project Master Database Model, TRW, Redondo Beach, CA, 1984.

Penedo, Maria H. and Stuckle, E.D., "PMDB - A Project Master Database for Software Engineering Environments," submitted for publication to the 8th International Conference on Software Engineering, London, England, August 1985.

APPENDIX 2

Some Thoughts on A Taxonomy for Software Engineering Objects

Leon G. Stucki

This short concept paper addresses the need for some type of taxonomy or information model within a software engineering environment. Such an information model is needed to identify and classify the different types of "software engineering objects" that a computer-aided software engineering environment will have to support. This in turn will help identify some of the high level requirements for information management.

For example, source code, execution scripts, test data, et. al. all suggest the need for simple files with configuration control/configuration management attributes. Other "software engineering objects", however, suggest the need for very powerful and efficient database capabilities. Structured specification concept and system models involving graphics, forms, as well as textual related subobjects suggest the need for powerful database management capabilities in addition to the normal relational operations (such as indexed or keyed fields and the ability to manipulate variable length text fields). In addition to the representational problems - configuration management/change control, backup, transaction logging, resource estimation and tracking, et. al. are also important considerations.

My major recommendation is that some extra time be allotted to allow for the development of a high level information model which will include as a minimum:

- A list of the most obvious "object types" to be managed by the system.

- A list of candidate data structures to be considered for representing the respective types of objects together with their associated attributes and values.

- A list of views and user perspectives to be supported.

- Proposed techniques for isolating the human interaction from the computational aspects of each type of tool as suggested in the companion "Human Engineering" paper should also be included in the model.

Software Engineering Objects	Generic Storage Model
Planning and Estimating	
Pert Charts	Structured Graphics
Gantt Charts	Structured Graphics
Task Spec Sheets	Forms
Documentation	
Documentation Plan	Structured Document
Formal Specification	Structured Document
Formal Design	Structured Document
User's Documentation	Structured Document
Test Plan	Structured Document
Test Procedures	Structured Document
Test Summary	Structured Document
Specification and Analysis	
Context Diagrams	Structured Graphics
Data Flow Diagrams	Structured Graphics
State Transition Diagrams	Structured Graphics
Spec Sheets	Forms
Rapid Prototype Systems	Interpretive System
Applications Generators	Interpretive System
Design	
Structure Charts	Structured Graphics
Program Design Languages	Sequential Units
Nassi Schneiderman Charts	Structured Graphics
database Design Models	Database Model
Spec Sheets	Forms
Database Schemas	Database Templates
Programming	
Source Code	Sequential Units
Object Code	Sequential Units
Libraries	Sequential Units
Build/Make Files	Sequential Units
Command Scripts	Sequential Units
Database Schemas	Database Templates
Debugging Scripts	Sequential Units
Test Data	Sequential Units
Test Scripts	Sequential Units
Test Results	Sequential Units
Integration and Testing	
Controlled Source Code	Sequential Units
Controlled Object Code	Sequential Units
Controlled Libraries	Sequential Units
Controlled Build/Make Files	Sequential Units
Controlled Command Scripts	Sequential Units
Controlled database Schemas	Database Templates
Controlled Test Data	Sequential Units
Controlled Test Scripts	Sequential Units
Controlled Test Results	Sequential Units

Table 1: Software Engineering Objects and Generic Storage Models

Analysis of Table 1 suggests that a data repository for a software engineering environment should be capable of effectively manipulating the following generic data types:

Structured Graphics - Where a given "structured graphics object"is made up of more than just a diagram. In fact, the concept really implies the existance (or potential existence) of a set of related diagrams, hierarchical derivations/relationships, and (object, attribute, value) relations. For example, support for a Data Flow Diagram "object" would include sufficient information to fully characterize all of its constituent "parts" or "subobjects" together with a mechanism for manipulating all of their corresponding attributes and values. Connectivity and decomposition information must also be supported. Using this generic storage model, a sophisticated editor can be built which will support real-time decomposition of graphic objects, rubberbanding of diagram subobjects as they are modified, and the specification of attribute values by filling in the blanks in appropriate "forms".

Structured Documentation - Where a given "structured documentation object" is made up of various "subobjects" such as sectional headings, chapter headings, subchapter headings, textual units, illustrations, tables, figures, etc. An editor can then support and maintain the structure of the document. Libraries of sample textual units can be developed and used to speed document preparation and maintenance. A fisheye browser feature can be included within the editor to provide improved contextual awareness. (E.g., Headings can be shown while detailed text is not for areas out of the center of the screen - giving a "fisheye" effect.)

Sequential Units - Where "sequential units" can be viewed as related sets of sequential information. Typical relationships include controlled storage and access to source code modules which make up a total program or subsystem. Another example relationship is the maintenance of release information by means of a set of master files and delta files capable of generating specific versions of "objects".

Database Model, Templates, and Forms - Where a specific set of database capabilities is provided. The database must be capable of all the normal relational type database operations. It should also have a forms mode for entry, query/modification, and report generation. For efficiency reasons it must support keyed access to various fields. Transaction logging and backup capability, security, submodel - record - field locking, and access rights are some of the other desirable characteristics of the database. Efficient implimentation of variable length fields will support the previously mentioned sequential units, structured documentation, and structured graphics "objects."

Interpretive Systems - Where a collection of "tools" are included which have the ability of deriving control information

from various design representations (e.g., State Transition Diagrams, Ada PDL, etc.) and generating operational skeleton code. This skeleton code can then be linked together with semantic specifications from the database for prototyping user interfaces with little or no real "coding" having been performed. Tools to synthesize "smart functional stubs" based on their input/output specs ("package definitions") and additional PDL info specifying linkages to libraries of existing code modules, models (or "packages") can also be provided.

Comments on the general architecture for the software engineering environment:

Conjecture:

Given the an ability to represent the above mentioned "objects" in an efficient manner, the primary architecture for the environment is centered around a very powerful editor linked with the database and capable of manipulating each of the above "objects." Analysis functions can be developed and stored as operations which can be applied to the objects stored in the database. (Basically, the architecture should result in a very powerful "visi-like" [what you see is what you get] capability for manipulating software engineering objects.)

DISTRIBUTION LIST FOR IDA PAPER P-1821

DoD

Col. Joe Greene — 10 copies
Director, STARS Joint Program Office
1211 Fern St., C-107
Arlington, VA 22202

Other

Mr. Jack C. Wileden — 5 copies
COINS Dept.
Lederly Graduate Research Center
University of Massachusetts
Amherst, MA 01003

Defense Technical Information Center — 2 copies
Cameron Station
Alexandria, VA 22314

CSED Review Panel

Dr. Dan Alpert, Director
Center for Advanced Study
University of Illinois
912 W. Illinois Street
Urbana, Illinois 61801

Dr. Barry W. Boehm
TRW Defense Systems Group
MS 2-2304
One Space Park
Redondo Beach, CA 90278

Dr. Ruth Davis
The Pymatuning Group, Inc.
2000 N. 15th Street, Suite 707
Arlington, VA 22201

Dr. Larry E. Druffel
Software Engineering Institute
Shadyside Place
480 South Aiken Av.
Pittsburgh, PA 15231

Dr. C.E. Hutchinson, Dean
Thayer School of Engineering
Dartmouth College
Hanover, NH 03755

Mr. A.J. Jordano
Manager, Systems & Software
Engineering Headquarters
Federal Systems Division
6600 Rockledge Dr.
Bethesda, MD 20817

Mr. Robert K. Lehto
Mainstay
302 Mill St.
Occoquan, VA 22125

Mr. Oliver Selfridge
45 Percy Road
Lexington, MA 02173

IDA

Gen. W.Y. Smith, HQ
Mr. Seymour Deitchman, HQ
Ms. Karen Webber, HQ
Dr. Jack Kramer, CSED
Dr. John Salasin, CSED
Dr. Robert Winner, CSED
Dr. Richard P. Morton 5 copies
Ms. Katydean Price, CSED 2 copies
IDA C&D Vault 3 copies

END

12-87

DTIC